Bandit Problems on Parallel Machines

Ryan Thomas Dunn

Thesis submitted for the degree of

Doctor of Philosophy

The University of Edinburgh

2005

Abstract

Glazebrook and Wilkinson (2000) exploited the primal-dual approach within achievable region methodology to investigate the classical multi-armed bandit problem on identical machines working in parallel. We follow this analysis by utilising and developing elements of the account, given by Bertsimas and Niño-Mora (1996), of the branching bandit model on a single machine cast in the achievable region framework, in order to extend Glazebrook and Wilkinson's work to more general models in the parallel machine environment. We obtain performance guarantees for a range of simple index based heuristic policies for models in which bandits compete for processing by machines of differing speeds and also where the number of available (identical) machines is a stochastic process. From these performance guarantees various forms of asymptotic optimality are established. Numerical studies allow insights concerning the degree of conservatism in the theoretical results.

Acknowledgements

I would like to express my sincere gratitude to my supervisor Professor Kevin Glazebrook for the belief, support and guidance shown throughout the course of my studies. His encouragement, patience and compassion were greatly appreciated.

I am grateful to Dr. Phil Ansell of the University of Newcastle Upon Tyne and to Richard Lumley for their helpful advice regarding computational work, also to my friends and family their continual emotional support.

I should also express my appreciation to the Engineering and Physical Sciences Research Council for supporting this work by means of a research studentship.

Declaration

I hereby declare that this thesis is my own work, prepared and completed with Professor Glazebrook as first supervisor.

Ryan Thomas Dunn.

Contents

List of Tables

Chapter 1

Introduction

1.1 Overview

The modelling, analysis and optimization of complex, stochastic service systems has been the focus of a substantial amount of research in recent times. One motivation behind this work has been the idea of optimal resource allocation, where the aim is to optimize, in a sequential manner, the allocation of effort between a number of competing projects. The outcome of each decision is uncertain and affects the situation and options available in the future. Examples include an industrial processor with jobs of different classes waiting to be processed and a server with a queue of customers of different types. Such problems seek to determine the best course of action in a decision problem, under the restrictions of limited resources. However, stochastic optimization in complex systems such as these, is technically very challenging.

This chapter gives a brief introduction to stochastic scheduling problems and some of the methods that have been used to analyse them. Following a short section on traditional dynamic programming formulations, the bandit problems of the thesis title are discussed. Specifically these are the discounted multi-armed bandit problem and the discounted branching bandit problem. An outline of the solution of the classical single machine versions of these problems is given, before recent approaches to the analysis of stochastic and dynamic scheduling problems are introduced. These methods, founded in mathematical programming, allow an alternative interpretation of the traditional dynamic programming based solutions of bandit problems on a single machine. A development of this approach which provides a framework for the analysis of systems

with a collection of machines working in parallel is then examined. Finally the structure of the thesis is outlined. Throughout this introductory chapter, problems are formulated and methods described using notation which will simplify later discussions.

1.2 Dynamic Programming

For the most part, stochastic and dynamic scheduling problems have been approached via dynamic programming (DP) formulations. The central idea behind DP is based on a principal of optimality due to Bellman (1957). The principle yields recursive equations for the associated value function of the problem, which occasionally can be solved analytically and an optimal policy determined. Often though, the generality of the optimality equations provides little insight into the structure of an optimal procedure. In principle at least, numerical solutions can be found for relatively small problems, though as a system becomes more complex, problems become computationally intractable.

For these reasons, developments in stochastic scheduling problems within the theoretical framework of DP formulations have, in the main, relied on heuristic methods. One of the simplest and most effective of these methods have been interchange arguments, which exploit the structure of the particular problem. An interchange argument establishes the optimality of a given policy by demonstrating that any alternate policy can be improved by swapping the order in which actions are carried out. As a *general* solution technique, this has been relatively inadequate due to the fact that it utilizes problem specific properties. One notable success though was the solution of the multi-armed bandit problem. This simple stochastic resource problem is a classic model which can be described as follows:

1.3 The Multi-armed Bandit Problem

There are B bandits or projects available for processing by a single machine or server of unit capacity. Each bandit b evolves under processing within the countable state space E_b. The state of the system is a B vector whose b^{th} component is the state of bandit b. Write $E = \bigcup_b E_b$ for the disjoint union of the individual bandit state spaces. At each instant in discrete time $t \in \mathbb{N}$, one of the bandits is chosen for processing. Should bandit

b be chosen for processing at time t when in state $i \in E_b$, then with probability P_{ij}^b it will be in state $j \in E_b$ at time $t+1$. The transition law for each bandit is Markovian and distinct bandits are assumed to evolve independently of each other. Bandits not chosen for processing at t remain stationary.

The classical multi-armed bandit problem has a discounted reward criterion ensuring total expected rewards over an infinite time horizon are finite. Rewards earned by bandit b are given by the positive valued reward vector $\mathbf{r}^b = \{r_i, i \in E_b\}$ and if bandit b in state $i \in E_b$ is chosen for processing at time t, a reward of $r_i \int_t^{t+1} e^{-\alpha s} ds$ is earned, where the constant $\alpha > 0$ is a discount rate. A scheduling policy u is a rule for determining how bandits should be allocated to the machine at each decision epoch. The set of admissible policies is denoted $\mathcal{U}$ and is restricted to the class of non-anticipative and non-idling controls. Hence decisions must not be based on future information and the machine/server must be busy as long as there is work present in the system. The total expected reward earned under u from initial state $\mathbf{k} = \{k(1), k(2), \ldots, k(B)\}$ can be expressed as

$$R^u(\mathbf{k}) = \sum_{i \in E} r_i x_i^u(\mathbf{k}), \tag{1.1}$$

where

$$x_i^u(\mathbf{k}) = E_u\Big\{ \int_0^\infty \chi_i(s) e^{-\alpha s} ds | \mathbf{k} \Big\}, \, i \in E, \tag{1.2}$$

represents the total expected discounted time spent working on a bandit in state i under policy u. For each $t \in \mathbb{N}$, $\chi_i(s)$, $t \leq s < t+1$, is given by

$$\chi_i(s) = \begin{cases} 1, & \text{if at decision epoch } t \text{ a bandit in} \\ & \text{state } i \text{ is chosen for processing,} \\ 0, & \text{otherwise.} \end{cases} \tag{1.3}$$

The objective is to find an optimal scheduling policy, within the class of admissible policies, to maximize the total expected discounted reward over an infinite time horizon. This can be expressed as the stochastic optimisation problem

$$R^{\text{opt}}(\mathbf{k}) = \sup_{u \in \mathcal{U}} R^u(\mathbf{k}). \tag{1.4}$$

Note that, the dependence upon the system's initial state $\mathbf{k}$ may be supressed when no confusion arises.

Since the formulation of this problem is in discrete time, the integral in (1.2) could be replaced by a sum. In traditional constructions of a multi-armed bandit problem, rewards are earned immediately when processing starts. Hence if a bandit in state $i \in E_b$ is chosen for processing at time t, under a traditional formulation, the reward earned is $r_i e^{-\tilde{\alpha}t}$ with discount rate $\tilde{\alpha} > 0$ and (1.2) expressed as a sum. To simplify computations however, an integral in (1.2) is preferred. By considering $\tilde{\alpha} = \alpha \ln\left[(1-e^{-\alpha})/\alpha\right]$ it follows that

$$\begin{aligned} e^{-\tilde{\alpha}t} &= \frac{e^{-\alpha t}(1-e^{-\alpha})}{\alpha} \qquad (1.5) \\ &= \int_t^{t+1} e^{-\alpha s} ds, \end{aligned}$$

hence giving an integral in (1.2).

The multi-armed bandit problem was formulated during the Second World War though it was not solved until a number of years later. Gittins and Jones (1974) used interchange arguments to solve the problem and established optimal policies determined by a collection of ranking indices. The original proof however was difficult to follow, consequently it was not until Gittins (1979), where a unified account of the theory behind the work was presented, that the result became more widely accepted. Whittle (1980) also gave a key account of the multi-armed bandit problem, offering an alternative DP based proof, briefly discussed in Section 1.3.1.

In the solution of the multi-armed bandit problem, an index is attached to each bandit state and effort is optimally allocated at each decision epoch to whichever bandit has the highest current index value. The indices concerned are known as Gittins indices, a term adopted by Whittle (1980). The procedure associated with the allocation of effort based on these indices is a Gittins index policy (or simply an index policy), denoted u_G.

1.3.1 Gittins Indices and Index Policies

Gittins' proof of the multi-armed bandit problem relied on interchange arguments based on a family of stopping times defined on the bandit process. Consider an individual bandit b, with initial state $i \in E_b$, processed continuously from time 0 onwards. Let τ be a positive valued stopping time, then define the index G_i, $i \in E_b$, as

$$G_i = \sup_{\tau>0} \frac{E\left[\sum_{j\in E_b} r_j \int_0^\tau \chi_j(s)\, e^{-\alpha s} ds \mid k(b) = i\right]}{1 - E(e^{-\alpha\tau})} \tag{1.6}$$

with $\chi_j(s)$ as in (1.3). From this definition, the index for state $i \in E_b$ can be interpreted as a maximum expected reward rate per unit of discounted time, accrued from the continuous processing of bandit b, when processing begins from state i at time 0.

Gittins proved that the indices corresponding to the states of a particular bandit depend only on the characteristics of that bandit, namely states, rewards and transition probabilities. This decomposition property allows a B-dimensional problem to be reduced to B one-dimensional problems. Hence a B-armed bandit problem is computationally equivalent to B one-armed bandit problems, meaning the computation of indices can be achieved via a number of smaller sub-problems, reducing the amount of computations required and making the addition/removal of bandits within the problem much simpler.

Gittins' result uncovered a remarkably simple structure to a complex problem, establishing the optimality of a priority index policy. Hence the members of the set of all bandit states E, may be ordered such that always choosing the bandit whose current state has highest Gittins index value will maximise the expected discounted reward to infinity. So it is optimal to give, at each decision time, higher priority to states with larger indices. Such policies are deterministic; since the ordering of states is fixed, stationary; since there is no explicit time dependence and Markovian; since choices made at time t depend only on the state of the system at time t. The nature of the Gittins index allows an insight into why the policy is optimal.

Whittle (1980) further clarified this result, offering an interpretation of the Gittins index as an equivalent retirement reward. Consider a single bandit b, with initial state $i \in E_b$ processed continuously from time 0. The permanently binding option of retiring from processing and receiving a retirement reward of $\rho \int_t^\infty e^{-\alpha s} ds$ is given at each decision epoch $t \in \mathbb{N}$ (providing retirement has not been taken at some time $s < t$). The Gittins index for state $i \in E_b$ is then the smallest value of ρ for which one is indifferent between retiring or continuing processing bandit b when in state i. By extending this result to the multi-armed bandit problem, Whittle obtained an expression for the value function under a Gittins index policy. Whittle then showed that this value function satisfies the DP optimality equations, which for discounted

problems, is a necessary and sufficient condition for optimality.

Gittins index policies offer efficient solutions both in theory and practice. They are relatively easy to construct and in principle straightforward to implement, though they may involve considerable switching between bandits. In many systems, excessive switching between bandits may not be practical and suboptimal heuristic policies will often be of interest. Glazebrook (1982) showed that the Gittins indices which define an optimal policy in the multi-armed bandit problem, can also play a role in evaluating the performance of suboptimal policies. Glazebrook has investigated different kinds of suboptimality, with his approach typically being to measure the consequences (in terms of reward earned) of breaking a necessary optimality condition in terms of the extent to which the condition has been broken. For example, in the multi-armed bandit case, the degree to which a suboptimal policy deviates from an index policy is measured by the aggregated difference, over all decision epochs, between the Gittins indices of the jobs chosen under the policy implemented and those of the optimal policy.

Developments of index theory within the framework of dynamic programming formulations have shown that index policies are optimal for a variety of dynamic and stochastic scheduling problems. Systems whose optimal controls can be characterized by a set of priority indices are called indexable by Bertsimas and Niño-Mora (1996). Extensions of the multi-armed bandit problem such as models incorporating a branching mechanism for processed projects have been used to model open systems in which new jobs arrive for processing.

The idea of branching was first introduced in the context of bandit problems by Glazebrook (1976) who showed that an index policy is optimal in a system where constraints on processing are expressed in an outforest structure. This procedure can be used to model systems with arrivals. Whittle (1981) worked on systems with arrivals, and important work on establishing the optimality of index policies for a general branching bandit problem was by Weiss (1988). The branching bandit is a versatile model. Systems that can be modelled as branching bandits include the multi-armed bandit problem defined earlier, as well as multiclass queues in discrete or continuous time. The discounted branching bandit model is described as follows:

1.4 The Branching Bandit Problem

A single machine or server must be allocated over time to jobs requiring processing. Jobs are classified in one of a finite number of classes, labelled $\{1, 2, \ldots, C\}$. The state of the system at time $t \in \mathbb{R}$ is $\mathbf{N}(t) = \{N_1(t), N_2(t), \ldots, N_C(t)\}$ and records the number of jobs within each class present and requiring service at t. If a class i job is chosen for processing at time $t \in \mathbb{R}$ then at time $t + \nu_i$, that class i job disappears to be replaced by $\mathbf{n}_i^t \equiv \{n_{i1}^t, n_{i2}^t, \ldots, n_{iC}^t\}$ jobs of classes $1, 2, \ldots, C$ respectively. The random service time ν_i and the random arrivals $\mathbf{n}_i^t$ associated with a given job class i, are random variables with an arbitrary joint distribution, independent of all other jobs. Decision times are $t = 0$ and the instants at which a job completes its service and some other job is present in the system. The processing of a class i job at time t earns a reward $r_i \int_t^{t+\nu_i} e^{-\alpha s} ds$ with $\alpha > 0$ a discount rate.

The objective is to find a scheduling policy to maximize the total expected discounted reward over an infinite time horizon. Admissible policies must be non-anticipative, non-idling and non-preemptive. Hence decisions must not be based on future information, the server must be busy as long as jobs are present in the system and once a job is in service, it must proceed until completion without interruption. The problem may be expressed as the stochastic optimization problem

$$R^{\text{opt}}(\mathbf{k}) = \sup_{u \in \mathcal{U}} \Big(\sum_{i \in E} r_i x_i^u(\mathbf{k}) \Big) \tag{1.7}$$

where

$$x_i^u(\mathbf{k}) = E_u \Big\{ \int_0^\infty \chi_i(t) e^{-\alpha t} dt | \mathbf{k} \Big\}, \, i \in E \tag{1.8}$$

and

$$\chi_i(t) = \begin{cases} 1, & \text{if a class } i \text{ job is processed at time } t \\ 0, & \text{otherwise.} \end{cases} \tag{1.9}$$

Hence, x_i^u represents the total expected discounted time for which class i jobs are processed under scheduling policy u. An optimal solution to the branching bandit problem attaches an index to each job class, the resulting index policy will, at each decision time, select the job with the largest current index for processing. Often, particularly when modelling multi-class queueing systems, branching bandit problems have an objective based on holding costs. A cost c_i is attached to each class i job present in the

system, $i \in E$ and is incurred per unit time that a class i job remains in the system. Notice though, that such cost minimisation problems can be formulated as an analogous reward maximisation problem.

1.5 The Achievable Region Approach

Recent analyses of dynamic and stochastic scheduling problems have established relationships between the strong structural properties of the solutions of certain classes of problem and the physical properties of the system. The so-called achievable region approach applies techniques founded in mathematical programming to solve stochastic optimization problems. In broad terms, the achievable region approach develops solutions by characterizing the space of all possible performances (the achievable region) of the system of interest and then optimizing the overall system-wide performance objective over this space. A control that achieves the optimal performance is then identified.

The multi-armed bandit problem and branching bandit problem outlined earlier are examples of systems where collections of jobs of different classes are available for processing by a single machine or server. The aim is to find a policy, within the class of admissible policies, that optimises some performance objective over the system. In both of these problems, an optimal policy is a static priority policy which at each decision epoch, gives higher priority to a job class with larger index. Index policies and policies based on Gittins indices are central throughout the thesis. The main ideas and methods behind the achievable region approach, provide an effective framework for tackling scheduling problems solved by priority index rules. Note however that the achievable region approach has been applied to systems other than those that are indexable, see for example Dacre, Glazebrook and Niño-Mora (1999).

In many stochastic scheduling problems, the objective to be optimised can be expressed as a linear combination of suitably defined performance measures. A performance vector $\mathbf{x}^u$ is associated with each control $u \in \mathcal{U}$, the set of admissible policies. The class i performance, x_i^u, is typically an expectation of some quantity related to class $i \in E$. For example in the branching bandit model in Section 1.4, x_i^u is the total expected discounted time for which class i jobs are processed under policy u.

As policies range over the admissible class, the performance vectors span the region

of achievable performance denoted $X = \{\mathbf{x}^u, u \in \mathcal{U}\}$. By identifying physical laws, known as conservation laws, describing the behaviour of the system under different scheduling policies and expressing them as linear constraints on the performance vectors, the performance region X may sometimes be constructed. Given a reward vector $\mathbf{r} = \{r_i, i \in E\}$, the stochastic optimisation problem can then be expressed as

$$\begin{aligned} Z^{\text{opt}} &= \sup_{u \in \mathcal{U}} \Big(\sum_{i \in E} r_i x_i^u \Big) \\ &= \max_{\mathbf{x} \in X} \Big(\sum_{i \in E} r_i x_i \Big). \end{aligned} \tag{1.10}$$

In certain cases, the performance space X can be identified exactly and shown to have a special structure, allowing efficient solutions of (1.10). The solution yielded in such cases, $\mathbf{x}^{\text{opt}}$, the performance optimizing Z^{opt}, maybe of a form making the identification of an optimal control u^{opt} relatively straightforward. Specifically, optimal controls are often strict priority policies based on some ordering of E, which give priority to job class i over job class j, if at each decision epoch, a class j job is selected only if no class i jobs are available for selection.

Foundational contributions to the achievable region approach by Coffman and Mitrani (1980) and Gelenbe and Mitrani (1980) applied mathematical programming ideas to the problem of scheduling service in a multiclass queueing system, with linear delay costs. Using conservation laws, they were able to show that the performance region can be described as a polyhedron. Federgruen and Groenevelt (1988a,1988b) showed that the polyhedron that defines the performance space in certain special cases of multiclass queues has a special structure. The form of the polyhedron is defined in polyhedral combinatorics as a polymatroid. Shanthikumar and Yao (1992) advanced the theory further, developing a formal definition of strong conservation laws (SCLs).

1.5.1 Strong Conservation Laws (SCLs)

A performance vector is said to satisfy SCLs if the total performance over all job classes is invariant under any admissible control and the minimal total performance over the job classes in any subset of job classes, $S \subset E$, is achieved by any absolute priority rule which gives jobs in S^c priority over those in S. More formally, a system satisfies SCLs

if there exists a performance vector $\mathbf{x}$ and a set function $b : 2^E \mapsto \mathbb{R}^+$ such that,

$$\sum_{i \in E} x_i^u = b(E), \qquad u \in \mathcal{U} \tag{1.11}$$

and

$$\sum_{i \in S} x_i^u \geq b(S), \qquad u \in \mathcal{U}, \tag{1.12}$$

where the right hand side of inequality (1.12) is attained by any control $u : S^c \rightarrow S$. This requirement is expressed by

$$\sum_{i \in S} x_i^u = b(S), \qquad \text{for } u : S^c \rightarrow S. \tag{1.13}$$

Shanthikumar and Yao (1992) demonstrated that when SCLs apply, the achievable region of performance is necessarily a polyhedron of polymatroidal type. Polymatroids arise as the convex hull of feasible solutions of special classes of linear programs (LPs) which can be solved by a greedy algorithm, unlike general LPs which are solved, for example, by the simplex method. Edmonds (1970) proved that a greedy algorithm solves the linear programming problem over a polyhedron for every linear objective function, if and only if the polyhedron is a polymatroid. Therefore optimization problems whose performance measures satisfy SCLs can be formulated as LPs over polymatroids and solved by a greedy algorithm.

Shanthikumar and Yao (1992) showed that a wide variety of multiclass queueing systems have associated performance measures which satisfy SCLs. They also proved that when the cost is linear in the performance, the optimal policy is a static priority rule. This follows since the optimal value of an LP is realised at some extreme point of its feasible region and by using the result of Edmonds (1970), outlined above, together with the fact that the vertices of a polymatroidal achievable region can be shown to be the performance vectors of the absolute priority rules. Such policies order the job classes $i \in E$ such that always choosing the job class with highest available priority will optimise the objective, effectively providing an index solution for systems satisfying SCLs. An example of such a rule is the so-called $c\mu$ rule for single server multiclass queueing systems which gives priority to job classes with the larger $c_i\mu_i$-value, where μ_i is the mean service time for jobs in class i. Many systems which do not satisfy SCLs however, are solved by priority policies. Bertsimas and Niño-Mora (1996) took the achievable

region approach decisively further forward, extending the work of Shanthikumar and Yao (1992) by introducing generalized conservation laws (GCLs) thereby providing a framework for tackling stochastic scheduling problems solved by priority index policies.

1.5.2 Generalized Conservation Laws (GCLs)

A performance vector is said to satisfy GCLs if there exist weights such that the total weighted performance over all job classes is invariant under any admissible policy $u \in \mathcal{U}$ and the minimum weighted performance over job classes in any subset $S \subset E$ is achieved by any static policy which gives jobs in S^c priority over those in S. More formally, a system satisfies GCLs if there exists a performance vector $\mathbf{x}$, a set function $b : 2^E \mapsto \mathbb{R}^+$ and a matrix $\mathbf{A} = (A_i^S)_{i \in S, S \subseteq E}$ satisfying $A_i^S > 0$ for $i \in S$, $S \subseteq E$, such that,

$$\sum_{i \in E} A_i^S x_i^u = b(E), \qquad u \in \mathcal{U} \tag{1.14}$$

and

$$\sum_{i \in S} A_i^S x_i^u \geq b(S), \qquad u \in \mathcal{U}, \tag{1.15}$$

where the right hand side of inequality (1.15) is attained by any control $u : S^c \to S$. This requirement is expressed by

$$\sum_{i \in S} A_i^S x_i^u = b(S), \qquad \text{for } u : S^c \to S. \tag{1.16}$$

Note that SCLs are the special form of GCLs obtained when $A_i^S = 1$, $i \in S$, $S \subseteq E$.

By Bertsimas and Niño-Mora (1996), when the performance vector $\mathbf{x}$ satisfies GCLs the feasible region of achievable performance is

$$X = \left\{ \mathbf{x} \in (\mathbb{R}^+)^N ; \sum_{i \in S} A_i^S x_i \geq b(S), S \subset E \text{ and } \sum_{i \in E} A_i^S x_i = b(E) \right\}. \tag{1.17}$$

The performance space X described by (1.17) is (the base of) an extended polymatroid. Bertsimas and Niño-Mora (1996) were able to show that the vertices of such a performance region are achieved by static priority scheduling policies. Optimization of a linear objective such as (1.10) over an extended polymatroid is solved by the so-called adaptive greedy algorithm (AG) developed by Tsoucas (1991) and presented in Section 1.5.3 below. Bertsimas and Niño-Mora (1996) showed that if a scheduling problem satisfies GCLs (1.14)-(1.16), then the associated performance region is an extended

polymatroid. Therefore if the performance objective to be optimised is linear, as in (1.10), then the AG provides an optimal solution. The solution provided is a priority index policy. Bertsimas and Niño-Mora (1996) proved that the allocation indices included in the output of the AG correspond exactly to Gittins indices and by considering the dual of the LPs, they were able to provide an algebraic characterization of Gittins indices as sums of optimal dual variables.

1.5.3 The Adaptive Greedy Algorithm AG(A, r)

The adaptive greedy algorithm requires inputs given by the matrix $\mathbf{A} = (A_i^S), i \in S, S \subseteq E$ arising in the GCLs together with the reward vector $\mathbf{r} = \{r_i, i \in E\}$ as in (1.10). The outputs from the algorithm are a vector $\boldsymbol{\pi}$, a permutation of E determining a policy which implements priorities among job types according to their indices, and $\bar{\mathbf{y}}$ a vector of dual variables from which the Gittins indices $G_i, i \in E$, are constructed.

Step 1. Set $S_1 = E$; set $\bar{y}^{S_1} = \max\Big\{r_i/A_i^E : i \in S_1\Big\}$;
pick $\pi_1 \in \operatorname{argmax}\Big\{r_i/A_i^E : i \in S_1\Big\}$;
set $G_{\pi_1} = \bar{y}^{S_1}$.

Step k. For $k = 2, \ldots, |E|$:
Set $S_k = S_{k-1} \setminus \{\pi_{k-1}\}$; set $\bar{y}^{S_k} = \max\Big\{\big[r_i - \sum_{j=1}^{k-1} A_i^{S_j} \bar{y}^{S_j}\big]/A_i^{S_k} : i \in S_k\Big\}$;
pick $\pi_k \in \operatorname{argmax}\Big\{\big[r_i - \sum_{j=1}^{k-1} A_i^{S_j} \bar{y}^{S_j}\big]/A_i^{S_k} : i \in S_k\Big\}$;
set $G_{\pi_k} = G_{\pi_{k-1}} + \bar{y}^{S_k}$.

Step $|\mathbf{E}| + 1$. For $S \notin \{S_1, \ldots S_{|E|}\}$, set $\bar{y}^S = 0$.

Bertsimas and Niño-Mora (1996) showed that although the permutation $\boldsymbol{\pi}$ is not necessarily uniquely determined by the input, the dual solution $\bar{\mathbf{y}}$ is, being independent of the way ties in the maximisations in the AG are broken. This consistency in $\bar{\mathbf{y}}$ translates to a similar consistency in the definition of the indices $G_i, i \in E$.

Bertsimas and Niño-Mora (1996) showed that the general class of branching bandit processes satisfy GCLs. Hence optimal solutions are priority index policies derived from AG. The class of branching bandit processes includes the discounted branching bandit problem and the discounted multi-armed bandit problem described earlier, as well as various multiclass queueing systems. The achievable region approach provides a unified

method for analysing these classical stochastic scheduling problems and has lead to a deeper understanding of their geometric structural properties.

Subsequent contributions have deployed the achievable region approach to analyse systems which come close to satisfying the GCL requirements, but fail to do so exactly. In such systems, the performance space X cannot be identified exactly and the conditions sufficient to establish the optimality of index policies fail narrowly. Methods for measuring the extent to which a system fails to satisfy GCLs and deriving performance guarantees for index policies in terms of such measures, have been developed using ideas based on the primal-dual structure of an LP. The primal-dual approach works by constructing both a heuristic solution to an appropriately defined primal LP, which is related to the stochastic optimization problem of interest, and also a feasible solution to the dual of a relaxation of it. Such ideas lie behind recent contributions by Dacre, Glazebrook and Niño-Mora (1999), Glazebrook and Garbe (1999), Glazebrook and Niño-Mora (1997,2001) and Glazebrook and Wilkinson (2000).

1.6 The Primal-dual Approach

The stochastic scheduling problems analysed in subsequent chapters of the thesis are such that the associated performance measures fail to satisfy GCLs exactly. Therefore Gittins index policies are not optimal, though they do come close to optimality. The primal dual approach provides a platform for analysing such systems by identifying linear constraints satisfied by the performance measure $\mathbf{x}^u$ under all $u \in \mathcal{U}$. For each subset $S \subseteq E$, define the non-negative quantity $\beta(S)$ to be the minimum value achievable by $\sum_{i \in S} A_i^S x_i^u$ under admissible policies. That is

$$\beta(S) = \inf_{u \in \mathcal{U}} \sum_{i \in S} A_i^S x_i^u, \quad S \subseteq E. \tag{1.18}$$

The constraints in (1.18) then define a polyhedron

$$P = \left\{ \mathbf{x} \in (\mathbb{R}^+)^N ; \sum_{i \in S} A_i^S x_i^u \geq \beta(S), \quad S \subseteq E \right\}, \tag{1.19}$$

that contains the performance space X.

If the optimal scheduling problem of interest is to choose an admissible policy to

maximise a linear performance objective, with optimal value Z^{opt}, then

$$Z^{\text{opt}} = \sup_{u \in \mathcal{U}} \Big\{ \sum_{i \in E} r_i x_i^u \Big\} \leq \max_{\mathbf{x} \in P} \Big\{ \sum_{i \in E} r_i x_i \Big\}, \tag{1.20}$$

where the right-hand side of (1.20) is an LP relaxation of the original problem. This relaxation, with associated optimal value Z^{LP}, seeks to maximise the objective function $\sum_{i \in E} r_i x_i$ over the polyhedron P. The inequality in (1.20) follows since $X \subseteq P$ and gives

$$Z^{\text{opt}} \leq Z^{LP}. \tag{1.21}$$

Utilizing standard LP theory, by weak duality the value corresponding to any feasible solution to the dual of this LP will bound Z^{LP} and hence will also bound Z^{opt}. If Z^D is the reward associated with such a dual feasible solution and Z^u is the reward associated with some heuristic control of interest $u \in \mathcal{U}$, then by the inferred inequalities

$$Z^u \leq Z^{\text{opt}} \leq Z^{LP} \leq Z^D, \tag{1.22}$$

it follows that the degree of reward suboptimality of u, $Z^{\text{opt}} - Z^u$, is bounded above by $Z^D - Z^u$. Hence by making appropriate choices of a feasible solution to the dual of the above LP and comparing this with the performance of u, it may be possible to obtain a performance guarantee for policy u.

The procedures used to analyse systems of interest where GCLs fail narrowly, exploit the novel account of Gittins indexation by Bertsimas and Niño-Mora (1996), given from the achievable region perspective. By Bertsimas and Niño-Mora (1996), the output of AG provides a dual solution $\bar{\mathbf{y}}$, which can be shown to be a feasible dual solution of a relaxation of the original problem.

The dual of the above LP relaxation is

$$\min \Big\{ \sum_{S \subseteq E} \mathrm{y}^S \beta(S) \Big\}, \tag{1.23}$$

subject to

$$\sum_{i \ni S} \mathrm{y}^S A_i^S \geq r_i, \quad \text{for } i \in E, \tag{1.24}$$

$$\mathrm{y}^S \leq 0, \quad S \subseteq E. \tag{1.25}$$

By Bertsimas and Niño-Mora (1996), $\bar{\mathbf{y}}$ satisfies the constraints (1.24) with equality, that is

$$\sum_{i \ni S} \bar{y}^S A_i^S = r_i, \qquad \text{for } i \in E, \tag{1.26}$$

which, as outlined below, enables expressions for Z^D and Z^u to be determined. Label the members of E, $\{1, 2, \ldots, |E|\}$, such that

$$G_{|E|} \geq G_{|E|-1} \geq \ldots \geq G_2 \geq G_1. \tag{1.27}$$

Identify $S_j = \{j, j-1, \ldots, 1\}$ as the subset of E of cardinality j with lowest Gittins indices, hence $S_{|E|} = E$, then from AG

$$\bar{y}^{S_j} = \begin{cases} G_j - G_{j+1}, & 1 \leq j \leq |E| - 1, \\ G_j, & j = |E|. \end{cases} \tag{1.28}$$

The dual value associated with $\bar{\mathbf{y}}$ is given by

$$Z^D = \sum_{S \subseteq E} \bar{\mathbf{y}}^S \, \beta(S), \tag{1.29}$$

and can therefore be expressed as

$$\begin{aligned} Z^D &= \sum_{j=1}^{|E|} \bar{y}^{S_j} \beta(S_j), \\ &= G_{|E|}\beta(E) - \sum_{j=1}^{|E|-1} (G_{j+1} - G_j)\beta(S_j). \end{aligned} \tag{1.30}$$

Expressions for the reward associated with some heuristic control of interest u also emerge from the structure of AG. Examining (1.26) and noting that the only subsets $S \subseteq E$ that correspond to non-zero dual variables and that contain i are $\{S_j \; ; \; i \leq j \leq |E|\}$, it follows that (1.26) can be re-written as

$$\begin{aligned} r_i &= \sum_{j=i}^{|E|} \bar{y}^{S_j} A_i^{S_j} \\ &= G_{|E|} A_i^E - \sum_{j=i}^{|E|-1} (G_{j+1} - G_j) A_i^{S_j}, \qquad i \in E. \end{aligned} \tag{1.31}$$

Therefore the reward associated with the heuristic u can be written as

$$\begin{aligned} Z^u &= \sum_{i=1}^{|E|} r_i x_i^u \\ &= G_{|E|} \sum_{i=1}^{|E|} A_i^E x_i^u - \sum_{i=1}^{|E|-1} \sum_{j=i}^{|E|-1} (G_{j+1} - G_j) A_i^{S_j} x_i^u \\ &= G_{|E|} \sum_{i \in E} A_i^E x_i^u - \sum_{j=1}^{|E|-1} (G_{j+1} - G_j) \sum_{i \in S_j} A_i^{S_j} x_i^u. \end{aligned} \tag{1.32}$$

A suboptimality bound for the heuristic u follows by comparing (1.30) and (1.32) and considering (1.22).

Dacre, Glazebrook and Niño-Mora (1999), Glazebrook and Garbe (1999), Glazebrook and Niño-Mora (1997,2001) and Glazebrook and Wilkinson (2000) utilized the primal-dual approach to develop performance guarantees for Gittins index policies in stochastic scheduling models where the single machine/server of the classical results is replaced by a collection of M identical machines working in parallel.

1.7 The Parallel Machine Environment

Considerable effort has been devoted to the analysis of stochastic scheduling problems on parallel machines. From this work, it has become clear that these problems are in general, much less tractable than their single machine counterparts. See for example Weber (1982), Weber, Varaiya and Walrand (1986) and Weiss (1990,1992,1995) for important contributions. The systems considered in subsequent chapters of the thesis include the discounted multi-armed bandit problem and the discounted branching bandit problem as outlined earlier but where processing is carried out on a collection of machines working in parallel. Within achievable region methodology, the technical challenge posed by these parallel machine systems lies in the fact that when the number of identical machines is two or more, GCLs are not satisfied exactly.

Bertsimas and Niño-Mora (1996) showed that GCLs are satisfied in the single machine case of branching bandit processes by using interchange arguments based on the single machine assumption. Index policies are optimal in this case since the theory of Section 1.5 applies. When there is more than one machine however, GCLs fail due to the complicating issue of how effectively controls utilize the full machine capacity. Significant, recent progress with parallel machine problems within multiclass queueing systems and the class of branching bandit processes, has exploited the primal-dual approach outlined in Section 1.6.

Glazebrook and Garbe (1999) utilized the primal-dual approach in analysing a general discounted multi-armed bandit problem on parallel machines. The formulation used discounting of the form $e^{-\alpha t}$, as outlined earlier, where $\alpha > 0$ is a discount rate. In this context, the total expected reward from the implementation of any policy $u \in \mathcal{U}$

is $O(1/\alpha)$. The policy examined, a Gittins index policy implemented in the parallel machine environment, chooses at each decision epoch, M bandits from the $B > M$ bandits available, whose associated Gittins indices are maximal. Glazebrook and Garbe (1999) showed that the reward from this policy comes within $O(1)$ of the optimal reward, subject only to the condition that the Markov chain modelling the evolution of each bandit is irreducible and has finite state space. This $O(1)$ suboptimality bound implies that for small enough discount rate α, the corresponding index policy is average reward optimal.

Glazebrook and Wilkinson (2000) further developed the tools of analysis used by Glazebrook and Garbe (1999), facilitating stronger results for the parallel machine version of the discounted multi-armed bandit problem. They examined a simple class of index based policies where the collection of available bandits is divided at time zero into M sub-collections and sub-collection m is then processed exclusively on machine m, $1 \leq m \leq M$. Glazebrook and Wilkinson (2000) made use of expressions of the form in (1.32) to formulate expressions for the expected rewards obtained from policies which make initial once for all allocations of bandits to machines. They showed that if this initial division is done appropriately, taking careful account of the index structure of the bandits, and if each machine operates an optimal policy (i.e. a Gittins index policy) for its allocated sub-collection, then the total reward earned from this approach comes within an $O(\alpha)$ quantity of optimality. An improved $O(\alpha)$ suboptimality bound for u_G, the Gittins index policy implemented in the parallel machine environment was also obtained under certain additional conditions.

These $O(\alpha)$ results imply that the corresponding limit policies (i.e. policies of equivalent structure but based upon limiting index values as $\alpha \rightarrow 0$) are 1-optimal and average overtaking optimal for multi-armed bandit problems on parallel machines. A more detailed account of the methods used to obtain the sub-optimality bounds (including a short section on asymptotic optimality) is given in Chapter 2.

1.8 Thesis Structure

Glazebrook and Wilkinson (2000) exploited the primal-dual approach within achievable region methodology to develop an analysis of the classical discounted multi-armed

bandit on identical machines working in parallel. The analysis produced performance guarantees for policies based on Gittins indices. From these guarantees, various forms of asymptotic optimality were established for the policies concerned. The aim of the thesis is to develop this work by extending these theoretical results to more general models.

Chapter 2 begins with a review of the analysis by Glazebrook and Wilkinson (2000) of the discounted multi-armed bandit problem on identical parallel machines. Details of the model and tools of analysis are presented, followed by an evaluation of the index based heuristic controls proposed. The derived suboptimality bounds on the rewards demonstrate strong performances from the policies considered. Chapter 2 concludes with a brief discussion of various forms of asymptotic optimality and a section focussing on the procedures used to carry out numerical investigations into the tightness of the performance guarantees developed and hence the conservatism of the theoretical results. The techniques presented in Chapter 2 provide the foundation for the analyses in Chapters 3 and 4 of bandit problems on (sometimes) non-identical machines working in parallel.

Chapter 3 considers general multi-armed bandit problems on parallel machines which may work at different speeds. Chapter 4 considers generalisations of the discounted branching bandit and the discounted multi-armed bandit problems on parallel machines to cases where the collection of machines available for processing is itself a stochastic process. Both chapters share the same basic structure, starting with a brief background reviewing previous work in the subject area before presenting the model and tools of analysis of the specific problems addressed. Performance guarantees are obtained for a range of controls based on Gittins indices and various forms of asymptotic optimality are established. Numerical studies investigate the performance of the approach and the effectiveness of the theoretical results.

Chapter 5 provides comments on the effectiveness of the tools of analysis presented in Chapters 3 and 4 to yield accurate conclusions. The individual models, tools of analysis and index-based heuristic controls are discussed and the theoretical results are assessed both in terms of their overall performance and the degree of concordance with the results of the numerical studies. A discussion of the scope for further research work in this area follows.

Publications

Two papers relating to some of the work in Chapters 3 and 4 have been accepted by leading journals. The details are as follows:

R.T. Dunn and K.D. Glazebrook,

Discounted multi-armed bandits on a collection of machines with varying speeds, Mathematics of Operations Research, to appear.

R.T. Dunn and K.D. Glazebrook,

The performance of index-based policies for bandit problems with stochastic machine availability,

Advances in Applied Probability, 33, 365-390, 2001.

Chapter 2

Multi-armed Bandit Problems on Identical Parallel Machines

2.1 Introduction

This chapter reviews the analysis by Glazebrook and Wilkinson (2000) of the discounted multi-armed bandit problem on a collection of identical machines working in parallel. The tools of analysis described, utilize and develop elements of the account given by Bertsimas and Niño-Mora (1996) of the branching bandit model on a single machine cast in the achievable region framework. The techniques presented provide a platform for the analyses of bandit problems on (sometimes) non-identical machines working in parallel which are discussed in Chapters 3 and 4.

The classic discounted multi-armed bandit problem is a special case of the branching bandit problem. Bertsimas and Niño-Mora (1996) established that branching bandit problems on a single machine satisfy generalized conservation laws (GCLs) as defined in Section 1.5.2. As such, optimal policies are priority index policies derived from the so-called adaptive greedy algorithm AG, given in Section 1.5.3. Glazebrook and Wilkinson (2000) considered an extension of this model to the parallel machine environment, where GCLs and therefore the conditions sufficient to establish the optimality of index policies, fail narrowly. By utilizing and developing elements of the account of Gittins indexation by Bertsimas and Niño-Mora (1996), Glazebrook and Wilkinson (2000) exploited the primal-dual approach, outlined in Section 1.6, to analyse index based policies on parallel machines.

By obtaining upper bounds for the optimal reward and deriving expressions for the total expected discounted reward earned when implementing policies based on Gittins indices (Section 1.3.1) performance guarantees for the given policies were established. The techniques used by Glazebrook and Wilkinson (2000) to obtain suboptimality bounds are introduced in this chapter and subsequently developed for the analysis of bandit problems on collections of (sometimes) non-identical machines in Chapters 3 and 4.

2.2 The Model

There are B bandits, which are available for processing by a collection of M identical machines, where $B > M \geq 1$. Each machine can process all B bandits. Each bandit b evolves under processing within finite state space E_b. Write $E = \bigcup_b E_b$ for the disjoint union of the individual bandit state spaces. At each instant in discrete time $t \in \mathbb{N}$, M bandits are chosen for processing. Should bandit b be chosen for processing at time t when in state $i \in E_b$, then with probability P_{ij}^b it will be in state $j \in E_b$ at time $t+1$. This transition law is Markovian and distinct bandits are assumed to evolve independently of each other. The $M - B$ bandits not chosen for processing at t remain stationary.

Bandit b's evolution under processing is determined by the irreducible one-step transition matrix $\mathbf{P}^b$, hence the associated Markov chain is positive recurrent. Such a chain would be observed in real time were bandit b to be processed without interruption. The finite state space of each bandit and the positive recurrence of the Markov chains modelling their evolution are important conditions in terms of examining the performance of scheduling policies as the discount rate approaches zero. This topic is discussed in Section 2.5.

The rewards earned by bandit b are given by the positive valued reward vector $\mathbf{r}^b = \{r_i, i \in E_b\}$ and under the discounted reward criterion, if bandit b in state $i \in E_b$ is chosen for processing at time t, a reward of $r_i \int_t^{t+1} e^{-\alpha s} ds$ is earned, where $\alpha > 0$ is a discount rate. Rewards are additive across machines and over time. The total expected reward earned under admissible policy $u \in \mathcal{U}$ from initial state $\mathbf{k} = \{k(1), k(2), \ldots, k(B)\}$ can be expressed as

$$R^u(\mathbf{k}) = \sum_{i \in E} r_i x_i^u(\mathbf{k}), \tag{2.1}$$

where

$$x_i^u(\mathbf{k}) = E_u\Big\{\int_0^\infty \chi_i(s)e^{-\alpha s}ds|\mathbf{k}\Big\}, i \in E, \tag{2.2}$$

represents the total expected discounted time spent working on a bandit in state i under policy u. For each $t \in \mathbb{N}$, $\chi_i(s)$, $t \leq s < t+1$, is given by

$$\chi_i(s) = \begin{cases} 1, & \text{if at decision epoch } t \text{ a bandit in} \\ & \text{state } i \text{ is chosen for processing,} \\ 0, & \text{otherwise.} \end{cases} \tag{2.3}$$

The goal is the analysis of the stochastic optimisation problem

$$R^{\text{opt}}(\mathbf{k}) = \sup_{u\in\mathcal{U}} \sum_{i\in E} r_i x_i^u(\mathbf{k}). \tag{2.4}$$

2.3 Tools of Analysis

Bertsimas and Niño-Mora (1996), in their analysis of the single machine case, showed that $x_i^u(\mathbf{k})$ as in (2.2) is a performance measure satisfying GCLs (1.14)-(1.16) with matrix $\mathbf{A} = (A_i^S), i \in S, S \subseteq E$, such that

$$A_i^S = \begin{cases} E\left(\int_0^{T_i^{S^c}} e^{-\alpha t}dt\right) / \left(\int_0^1 e^{-\alpha t}dt\right), & i \in S, \\ 0, & i \notin S, \end{cases} \tag{2.5}$$

or equivalently,

$$A_i^S = \begin{cases} \left\{1 - E\left(e^{-\alpha T_i^{S^c}}\right)\right\} / (1 - e^{-\alpha}), & i \in S, \\ 0, & i \notin S. \end{cases} \tag{2.6}$$

Define the random variables $T_i^{S^c}$ as follows: fix $i \in E_b$ and subset $S \subseteq E$ with $i \in S$. Consider bandit b initially in state i, being processed continuously by a single machine. Under such processing, bandit b evolves as a Markov chain according to the one-step transition matrix $\mathbf{P}^b$. Write $T_i^{S^c}$ for the first time at or after time 1 at which b enters S. It follows from the assumed irreducibility and hence positive recurrence of the Markov chain, that all positive moments of $T_i^{S^c}$ are finite.

By utilizing the primal dual approach, outlined in section 1.6, Glazebrook and Wilkinson (2000) derived expressions for the rewards $\{r_i, i \in E\}$ in terms of the quantities A_i^S and the Gittins indices G_i, $i \in E$. These expressions follow from the structure

of AG described in section 1.5.3 whose inputs are the matrix $\mathbf{A}$ and reward vector $\mathbf{r} = \{r_i, i \in E\}$ and whose outputs include the set of all non-negative Gittins indices. Recall from Chapter 1 the labelling of members of E according to the ordering of these indices, $\{1, 2, \ldots, |E|\}$, such that

$$G_{|E|} \geq G_{|E|-1} \geq \ldots \geq G_2 \geq G_1 \tag{2.7}$$

and the definition of $S_j = \{j, j-1, \ldots, 1\}$ as the subset of E with j lowest indices. Then following (1.31), the rewards $\{r_i, i \in E\}$ may be expressed as

$$\begin{aligned} r_i &= G_{|E|} A_i^E - \sum_{j=i}^{|E|-1} (G_{j+1} - G_j) A_i^{S_j}, \\ &= G_{|E|} - \sum_{j=i}^{|E|-1} (G_{j+1} - G_j) A_i^{S_j}, \qquad i \in E. \end{aligned} \tag{2.8}$$

Equation (2.8) uses the fact that $A_i^E = 1$, $i \in E$, which follows from (2.6) using the fact that $T_i^{E^c} = 1$.

By utilizing (2.8), along the lines of (1.32), expressions for the expected reward earned when implementing policy $u \in \mathcal{U}$ can be developed as follows:

$$\begin{aligned} R^u(\mathbf{k}) &= \sum_{i \in E} r_i x_i^u(\mathbf{k}), \\ &= G_{|E|} \sum_{i \in E} x_i^u(\mathbf{k}) - \sum_{j=1}^{|E|-1} (G_{j+1} - G_j) \sum_{i \in S_j} A_i^{S_j} x_i^u(\mathbf{k}) \qquad (2.9) \\ &\equiv G_{|E|} \left(\frac{M}{\alpha} \right) - \sum_{j=1}^{|E|-1} (G_{j+1} - G_j) A^u(S_j, \mathbf{k}), \qquad (2.10) \end{aligned}$$

where (2.10) follows from (2.9) by using the notational shorthand

$$A^u(S_j, \mathbf{k}) = \sum_{i \in S} A_i^S x_i^u(\mathbf{k}) \tag{2.11}$$

and the fact that, from (2.2), for all controls $u \in \mathcal{U}$,

$$\begin{aligned} \sum_{i \in E} x_i^u(\mathbf{k}) &= \int_0^\infty M e^{-\alpha s} ds \\ &= \frac{M}{\alpha}. \end{aligned}$$

An upper bound for the reward associated with an optimal policy is an immediate consequence of (2.10). By introducing

$$A(S, \mathbf{k}) = \inf_{u \in \mathcal{U}} A^u(S, \mathbf{k}), \tag{2.12}$$

it follows that

$$R^{\text{opt}}(\mathbf{k}) \leq G_{|E|}\left(\frac{M}{\alpha}\right) - \sum_{j=1}^{|E|-1}(G_{j+1} - G_j)A(S_j, \mathbf{k}), \tag{2.13}$$

for all initial states $\mathbf{k}$.

By Bertsimas and Niño-Mora (1996), in the single machine case, (2.13) can be shown to be satisfied with equality, since when $M = 1$, GCLs (1.14)-(1.16) are satisfied and therefore for each $S \subseteq E$, the infimum in (2.12) is achieved by any policy $u : S^c \to S$. Recall that a Gittins index policy u_G, in the single machine case, will at each decision epoch, schedule for processing, whichever bandit has the highest current index value. By (2.7), such a decision process is consistent with the choosing bandits according to the priority ordering $|E| \to |E| - 1 \to \ldots \to 2 \to 1$ and therefore $u_G : S_j^c \to S_j$ for all j. Hence when $M = 1$, a Gittins index policy is optimal.

A Gittins index policy u_G, when implemented in the parallel machine environment, $M > 1$, chooses at each decision epoch, the M bandits whose associated Gittins indices are maximal. In this case u_G does not necessarily achieve the infimum in (2.12). Glazebrook and Wilkinson (2000) analysed the performance of u_G and other policies based on Gittins indices when implemented in the parallel machine environment. Their main tool of analysis was the following suboptimality bound, which is an immediate consequence of (2.10) and (2.13).

$$R^{\text{opt}}(\mathbf{k}) - R^u(\mathbf{k}) \leq \sum_{j=1}^{|E|-1}(G_{j+1} - G_j)\big\{A^u(S_j, \mathbf{k}) - A(S_j, \mathbf{k})\big\}. \tag{2.14}$$

The practical challenge in obtaining suboptimality bounds for given policies, involves deriving expressions for the associated quantities $A^u(S, \mathbf{k})$ and $A(S, \mathbf{k})$ for $S \subseteq E$, or tight bounds on them. In Glazebrook and Wilkinson (2000), a key step in obtaining bounds on the degree of reward suboptimality is the development of lower bounds for the quantities $A(S, \mathbf{k})$. These bounds are given in Lemma 1 and used in Section 2.4 for the analysis of a simple class of index based policies.

Recall that the members of $E = \bigcup_b E_b$ are numbered in decreasing order of their Gittins indices such that

$$G_{|E|} \geq G_{|E|-1} \geq \ldots \geq G_2 \geq G_1.$$

Write

$$j(b) = \min\{j;\ j \in E_b\},\ 1 \leq b \leq B,$$

for the minimum index state for bandit b and number the bandits such that

$$j(1) > j(2) > \ldots > j(B). \tag{2.15}$$

Then write

$$B_j = \Big\{ b \in B; E_b \cap S_j = \emptyset \Big\}, \quad 1 \leq j \leq |E|,$$

for the collection of bandits whose state spaces have no intersection with the set S_j and write

$$\mu_j = |B_j|, \quad 1 \leq j \leq |E|.$$

When no ties exist between index values, μ_j is the number of bandits, all of whose indices exceed G_j. Notice that by the definition of the quantities involved,

$$j \leq j(m) - 1 \iff \mu_j \geq m.$$

Therefore $j \leq j(M) - 1$ implies $\mu_j \geq M$. In this situation, there exist admissible controls which never schedule members of S_j. Plainly under such controls, it must be true that for all initial states $\mathbf{k}$,

$$A^u(S_j, \mathbf{k}) = 0$$

and hence

$$A(S_j, \mathbf{k}) = 0.$$

Consider now the case $0 \leq \mu_j \leq M - 1$. Let bandit $b \notin B_j$ and use $T\{k(b), S_j^c\}$ to denote the first occasion at or after time 0 at which bandit b with initial state $k(b)$ escapes S_j^c. Then write

$$T(\mathbf{k}, S_j^c) = \sum_{b \notin B_j} T\{k(b), S_j^c\}, \tag{2.16}$$

for the total amount of processing required for all bandits outside of B_j to escape S_j^c (when operating a control giving priority to S_j^c over S_j).

Lemma 1 (Glazebrook and Wilkinson (2000))

If $0 \leq \mu_j \leq M - 1$, then

$$A(S_j, \mathbf{k}) \geq \frac{(M - \mu_j)}{\alpha} E\Big(\exp\{-\alpha T(\mathbf{k}, S_j^c)/(M - \mu_j)\} \Big). \tag{2.17}$$

If $\mu_j \geq M$ then

$$A(S_j, \mathbf{k}) = 0. \tag{2.18}$$

Proof

The proof of (2.18) is trivial and is contained in the above text. Hence we suppose that $0 \leq \mu_j \leq M-1$ and proceed to prove (2.17). Fix policy u and denote by $\tau_{i,l}$, the time of the l^{th} occasion upon which control u processes bandit b in state $i \in E_b \cap S_j$. With each $\tau_{i,l}$ is associated a random variable $T_{i,l}^{S_j^c}$ which is the length of the subsequent excursion of bandit b into S_j^c. For a given i, the $T_{i,l}^{S_j^c}$ are independent and identically distributed and share the distribution of $T_i^{S_j^c}$ defined following (2.6). The aim is to find lower bounds for the quantities

$$A(S, \mathbf{k}) = \inf_{u \in \mathcal{U}} A^u(S, \mathbf{k}) = \inf_{u \in \mathcal{U}} \sum_{i \in S} A_i^S x_i^u, \quad S \subseteq E.$$

Now from the definition of the quantities involved,

$$\begin{aligned}
A(S_j, \mathbf{k}) &= \inf_{u \in \mathcal{U}} \sum_{i \in S_j} A_i^{S_j} E_u \Big(\sum_{l=1}^{\infty} \int_{\tau_{i,l}}^{\tau_{i,l}+1} e^{-\alpha t} dt \,\Big|\, \mathbf{k} \Big) \\
&= \inf_{u \in \mathcal{U}} \sum_{i \in S_j} \frac{E\Big(\int_0^{T_i^{S_j^c}} e^{-\alpha t} dt \Big)}{\Big(\int_0^1 e^{-\alpha t} dt \Big)} E_u \Big(\sum_{l=1}^{\infty} e^{-\alpha \tau_{i,l}} \int_0^1 e^{-\alpha t} dt \,\Big|\, \mathbf{k} \Big) \\
&= \inf_{u \in \mathcal{U}} \sum_{i \in S_j} E_u \Big(\sum_{l=1}^{\infty} e^{-\alpha \tau_{i,l}} \int_0^{T_{i,l}^{S_j^c}} e^{-\alpha t} dt \,\Big|\, \mathbf{k} \Big) \\
&= \inf_{u \in \mathcal{U}} \sum_{i \in S_j} E_u \Big(\sum_{l=1}^{\infty} \int_{\tau_{i,l}}^{T_{i,l}^{S_j^c} + \tau_{i,l}} e^{-\alpha t} dt \,\Big|\, \mathbf{k} \Big). \qquad (2.19)
\end{aligned}$$

Consider bandit b chosen for processing at time $\tau_{i,l}$. The subsequent scheduling of bandit b until it next enters S_j will be delivered during $[\tau_{i,l}, \tau_{i,l} + T_{i,l}^{S_j^c})$ at the earliest. Notice also that once a bandit has visited S_j for the first time, subsequent visits will alternate with (possibly null) excursions into S_j^c. Hence from (2.19), the following inequality can be inferred;

$$A(S_j, \mathbf{k}) \geq \inf_{u \in \mathcal{U}} \sum_{b=1}^{B} \sum_{m=1}^{M} E_u \Big(\int_0^{\infty} I_{bm}(t) e^{-\alpha t} dt \Big) \qquad (2.20)$$

where

$$I_{bm}(t) = \begin{cases} 1, & \text{if bandit } b \text{ is processed at time } t, \text{ by machine } m, \\ & \text{having paid its first visit to } S_j \text{ at some time } s \leq t, \\ 0, & \text{otherwise.} \end{cases}$$

The right hand side of (2.20) is now bounded from below by relaxing the minimisation problem. The relaxation used takes the form of a single machine scheduling problem. Denote by the pair (m,t), $1 \leq m \leq M$, $t \in \mathbb{N}$, the decision opportunity afforded on machine m at time t in the parallel machine problem. Consider the mapping onto $\mathbb{N}$ given by

$$(m,t) \longrightarrow \phi(m,t) \equiv Mt + (m-1)$$

in which this decision opportunity is thought to occur on a single machine at time $\phi(m,t)$. The minimisation in (2.20) is relaxed by considering a suitable choice of admissible controls $\mathcal{U}'$ for the single machine problem. The admissible controls $\mathcal{U}'$ are non-anticipative and non-idling with the additional feature that the bandits in B_j whose state spaces have no intersection with S_j, may only be scheduled at certain decision epochs. The idea here is that these μ_j bandits, numbered $1, 2, \ldots, \mu_j$, can never contribute to the expression on the right hand side of (2.19) and hence should be scheduled whenever possible. In the single machine relaxation, policy $u' \in \mathcal{U}'$ can only schedule bandit $b \in \{1, 2, \ldots, \mu_j\}$ at times $\phi(b,t)$, $t \in \mathbb{N}$. At all other epochs, policy $u' \in \mathcal{U}'$ makes a free choice from bandits in B_j^c. Rewards accruing in the single machine problem at decision epoch $\phi(m,t)$ under $u' \in \mathcal{U}'$ attract discounting equal to that at time t in the original parallel machine problem.

Control $u' \in \mathcal{U}'$ will allocate bandits to machines in numerical order (for some suitable numbering of machines) at successive time points with repeat allocations allowed, subject to the requirement that bandit $b \in \{1, 2, \ldots, \mu_j\}$ may only be allocated to machine b. Note that this is equivalent to building a (possibly non-admissible) policy for the parallel machine problem. Hence under the single machine relaxation, it follows from (2.20) that,

$$A(S_j, \mathbf{k}) \geq \inf_{u' \in \mathcal{U}'} \sum_{b=1}^{B} \sum_{m=1}^{M} \sum_{t=0}^{\infty} E_{u'} \left[\left\{ \int_t^{t+1} e^{-\alpha s} ds \right\} \left\{ \int_{\phi(m,t)}^{\phi(m,t)+1} I_b(s) ds \right\} \Big| \mathbf{k} \right], \quad (2.21)$$

where

$$I_b(s) = \begin{cases} 1, & \text{if bandit } b \text{ is processed at time } s, \text{ having} \\ & \text{paid its first visit to } S_j \text{ at some time } v \leq s, \\ 0, & \text{otherwise.} \end{cases}$$

Obtaining policies which achieve the infimum in the single machine problem in (2.21) with admissible controls $\mathcal{U}'$ is straightforward. Firstly since bandits $b \in \{1, 2, \ldots, \mu_j\}$

never visit S_j, they will not contribute to the objective on the righthand side of (2.21). Hence, since the function $e^{-\alpha s}$ is decreasing in s, any u' obtaining the infimum in (2.21) must choose these bandits whenever possible i.e. bandit $b \in \{1, 2, \ldots, \mu_j\}$ will be scheduled at all qualifying epochs $\phi(b,t)$, $t \in \mathbb{N}$. Hence (2.21) can be replaced by

$$A(S_j, \mathbf{k}) \geq \inf_{u' \in \mathcal{U}'} \sum_{b=\mu_j+1}^{B} \sum_{m=\mu_j+1}^{M} \sum_{t=0}^{\infty} E_{u'} \left[\left\{ \int_t^{t+1} e^{-\alpha s} ds \right\} \left\{ \int_{\phi(m,t)}^{\phi(m,t)+1} I_b(s) ds \right\} \Big| \mathbf{k} \right]. \tag{2.22}$$

The righthand side of (2.22) is a minimisation involving the $B - \mu_j$ bandits in B_j^c, all of which can enter S_j. This single machine problem is easily solved. It follows from a simple pairwise interchange argument, based on the fact that the function $e^{-\alpha s}$ is decreasing in s, that a minimising $u' \in \mathcal{U}'$ will make all free choices at decision epochs $\{\phi(m,t);\, \mu_j + 1 \leq m \leq M;\, t \in \mathbb{N}\}$ by giving priority to S_j^c over S_j.

Recall that $T(\mathbf{k}, S_j^c)$ is the total amount of processing required for all bandits $b \in B_j^c$ to escape S_j^c when operating a control enforcing the priority $S_j^c \rightarrow S_j$. Write

$$T(\mathbf{k}, S_j^c) = \left\lceil \frac{T(\mathbf{k}, S_j^c)}{(M - \mu_j)} \right\rceil (M - \mu_j) + R$$

where $\lceil x \rceil$ is the integer part of x and $0 \leq R \leq M - \mu_j - 1$. A simple calculation yields

$$\begin{aligned} A(S_j, \mathbf{k}) &\geq E\Bigg((M - \mu_j - R) \int_{\left\lceil \frac{T(\mathbf{k}, S_j^c)}{(M-\mu_j)} \right\rceil}^{\infty} e^{-\alpha s} ds + R \int_{\left\lceil \frac{T(\mathbf{k}, S_j^c)}{(M-\mu_j)} \right\rceil + 1}^{\infty} e^{-\alpha s} ds \Bigg) \\ &= \frac{\left[(M - \mu_j - R) + R e^{-\alpha} \right]}{\alpha} E\Big[\exp\Big(- \alpha \big\lceil T(\mathbf{k}, S_j^c)/(M - \mu_j) \big\rceil \Big) \Big] \\ &\geq \frac{(M - \mu_j)}{\alpha} E\Big(\exp\left\{ - \alpha T(\mathbf{k}, S_j^c)/(M - \mu_j) \right\} \Big) \end{aligned}$$

as required.

The lower bound presented in Lemma 1 was developed by Glazebrook and Wilkinson (2000) and utilised in the assessment of heuristic policies for the multi-armed bandit problem on identical parallel machines. The following section gives details of the class of index based controls proposed by Glazebrook and Wilkinson (2000).

2.4 The evaluation of a class of index based controls

Consider a simple class of index based policies, which at time zero divide the collection of B available bandits into M sub-collections. Denote sub-collection m by B_m, $1 \leq m \leq M$. The bandits in B_m are then processed exclusively on machine m, $1 \leq m \leq M$, from time zero onwards. By the classical result of Gittins and Jones (1974), since Gittins index policies are optimal for the single machine problem, the total expected reward from a given partition will be optimised by using a Gittins index ordering $|E| \rightarrow |E|-1 \rightarrow \ldots \rightarrow 2 \rightarrow 1$ to schedule the bandits in B_m on machine m, $1 \leq m \leq M$. Use the notation $u_G(\mathbf{B})$ for any such policy. Glazebrook and Wilkinson (2000) developed a partition of bandits for which the performance is very close to optimal.

Recall the numbering of bandits in (2.15). Denote by $\mathbf{B}' = \{B'_m, 1 \leq m \leq M\}$, the partition whose associated policy $u_G(\mathbf{B}')$ allocates each bandit b, $1 \leq b \leq M-1$, to a single machine and once these $M-1$ single bandit assignments have been made, the bandits $\{M, M+1, \ldots, B\}$ are all assigned to the remaining machine. Since the machines are identical, the specification of how bandits are allocated to machines is immaterial. It is assumed, without loss of generality, that bandit m, with initial state $k(m)$, is allocated to machine m, $1 \leq m \leq M-1$ and bandits $\{M, M+1, \ldots, B\}$ are allocated to machine M. In what follows, for ease of notation, the initial state of the collection of bandits allocated to machine M, $\mathbf{k}(M)$, is written as $k(M)$.

In order to obtain sub-optimality bounds, as in (2.14), for policy $u_G(\mathbf{B}')$, access to the quantities $A^{u_G(\mathbf{B}')}(S, \mathbf{k})$, $S \subseteq E$, is required. Use $A_m^{u_G(\mathbf{B}')}\{S, k(m)\}$ to denote the contribution to $A^{u_G(\mathbf{B}')}(S, \mathbf{k})$ from the processing on machine m under control $u_G(\mathbf{B}')$. By the structure of $u_G(\mathbf{B}')$, no job type $j < j(m)$ will ever be processed by machine m, $1 \leq m \leq M$, since a job of type $j(m)$ will always be preferred. Hence

$$A_m^{u_G(\mathbf{B}')}\{S_j, k(m)\} = 0, \qquad j < j(m),\ 1 \leq m \leq M. \tag{2.23}$$

Now since each machine operates a Gittins index policy on the bandits allocated to it, jobs are chosen such that $S_j^c \rightarrow S_j$. Hence, for machine m, the first contribution to $A_m^{u_G(\mathbf{B}')}\{S, k(m)\}$ will occur at time $T\{k(m), S_j^c\}$, the first occasion at which a job in S_j is chosen. Following this epoch, excursions to S_j^c will alternate with allocations to

S_j. Hence, utilising the notation in the proof of Lemma 1 along with

$$E_{B_m} = \bigcup_{b \in B_m} E_b,$$

it follows that

$$A_m^{u_G(\mathbf{B}')}\{S, k(m)\} = \sum_{i \in S_j \cap E_{B_m}} E_{u_G(\mathbf{B}')}\Big(\sum_{l=1}^{\infty} \int_{\tau_{i,l}}^{T_{i,l}^{S_j^c} + \tau_{i,l}} e^{-\alpha t} dt |\, k(m) \Big) \tag{2.24}$$

$$= \frac{1}{\alpha} E\Big(\exp[-\alpha T\{k(m), S_j^c\}] \Big), \qquad j \geq j(m),\ 1 \leq m \leq M. \tag{2.25}$$

Now observe that when $0 \leq \mu_j \leq M-1$, the allocation of bandits to machines made under $u_G(\mathbf{B}')$ implies that

$$A^{u_G(\mathbf{B}')}(S, \mathbf{k}) = \sum_{m=\mu_j+1}^{M} A_m^{u_G(\mathbf{B}')}\{S, k(m)\}.$$

This yields

$$A^{u_G(\mathbf{B}')}(S, \mathbf{k}) = \frac{1}{\alpha} \sum_{m=\mu_j+1}^{M} E\Big(\exp[-\alpha T\{k(m), S_j^c\}] \Big), \quad j \geq j(m). \tag{2.26}$$

Utilising Lemma 1 and (2.26) within (2.14), an expression for a suboptimality bound for the policy $u_G(\mathbf{B}')$ is obtained as follows;

$$R^{\text{opt}}(\mathbf{k}) - R^{u_G(\mathbf{B}')} \leq \frac{1}{\alpha} \sum_{j=1}^{|E|-1} (G_{j+1} - G_j) \Bigg\{ \Bigg[\sum_{m=\mu_j+1}^{M} E\Big(\exp\big[-\alpha T\{k(m), S_j^c\}\big] \Big) \Bigg] - (M - \mu_j)\, E\Big(\exp\big\{ -\alpha T(\mathbf{k}, S_j^c)/(M-\mu_j) \big\} \Big) \Bigg\}. \tag{2.27}$$

By considering

$$e^{-\alpha t} = 1 - \alpha t + \frac{(\alpha t)^2}{2!} + O(\alpha^3)$$

and noting that

$$\sum_{m=\mu_j+1}^{M} T\{k(m), S_j^c\} = T(\mathbf{k}, S_j^c), \quad j(m) \leq j \leq |E| - 1,$$

Glazebrook and Wilkinson (2000) exploited the existence all positive moments of the random variables concerned to deduce that the reward earned from policy $u_G(\mathbf{B}')$ comes with $O(\alpha)$ of optimality, i.e.

$$R^{\text{opt}}(\mathbf{k}) - R^{u_G(\mathbf{B}')} \leq O(\alpha). \tag{2.28}$$

Comment

Glazebrook and Wilkinson (2000) also analysed a Gittins index policy u_G implemented in the parallel machine environment. Under u_G, at each decision epoch, M bandits whose current Gittins indices are maximal are chosen for processing. Ties are broken such that the priorities $|E| \rightarrow |E|-1 \rightarrow \ldots \rightarrow 2 \rightarrow 1$ are respected. Glazebrook and Wilkinson (2000) limited their analysis to models under which certain additional structural conditions hold. The classes of model analysed include the case in which all bandits are identical. By using methods and calculations similar to those outlined above, Glazebrook and Wilkinson (2000) proved the following result:
If

$$G_{j(n)} = G_{j(n+1)}, \quad 1 \leq n \leq M-1, \tag{2.29}$$

then

$$R^{\text{opt}}(\mathbf{k}) - R^{u_G} \leq O(\alpha). \tag{2.30}$$

2.5 Asymptotic optimality

The derivation of bounds on the degree of reward suboptimality of index based controls is a key feature of this and subsequent chapters of the thesis. Model assumptions guarantee that the expected rewards $R^{\text{opt}}(\mathbf{k})$ and $R^{u}(\mathbf{k})$, for all policies u considered, are $O(1/\alpha)$. The majority of suboptimality bounds obtained are proved to be $O(1)$ or $O(\alpha)$. See for example the bound in (2.28). A natural step in the analysis is the exploration of what these results imply when the limit $\alpha \rightarrow 0$ is taken. This includes the examination of the limiting forms of the policies discussed.

2.5.1 n-discount optimality

Various forms of asymptotic optimality are claimed for the limit policies examined and a central notion is that of n-discount optimality. Puterman (1994) gives a full discussion of this and other forms of asymptotic optimality set in the context of Markov decision processes (MDPs). A control u is said to be n-discount optimal if

$$\lim_{\alpha \rightarrow 0} \alpha^{-n} \left\{ R^{\text{opt}}(\mathbf{k}, \alpha) - R^{u}(\mathbf{k}, \alpha) \right\} = 0; \tag{2.31}$$

where in (2.31), the notations $R^{\text{opt}}(\mathbf{k})$ and $R^u(\mathbf{k})$ are expanded to express their dependence on α.

Special cases of n-discount optimality are **(-1)-discount optimality**, which holds for policy $u \in \mathcal{U}$ if

$$\lim_{\alpha \to 0} \alpha \{ R^{\text{opt}}(\mathbf{k}, \alpha) - R^u(\mathbf{k}, \alpha) \} = 0$$

and **0-discount optimality**. A policy $u \in \mathcal{U}$ is 0-discount optimal if

$$\lim_{\alpha \to 0} \{ R^{\text{opt}}(\mathbf{k}, \alpha) - R^u(\mathbf{k}, \alpha) \} = 0.$$

In general, n-discount optimality implies m-discount optimality for all m < n.

2.5.2 Finite state and action models

For discounted MDPs with finite state and action space, the classical theory and results of Blackwell (1962), Veinott (1966) and Denardo and Miller (1968) apply. By using a (partial) Laurent series expansion in terms of the discount rate α, the expected discounted reward for stationary policy u may be written

$$R^u(\mathbf{k}, \alpha) = \frac{R_1^u(\mathbf{k})}{\alpha} + R_2^u(\mathbf{k}) + \alpha R_3^u(\mathbf{k}) + O(\alpha^2), \tag{2.32}$$

where $R_1^u(\mathbf{k})$ is the usual average reward per unit time for u. See Blackwell (1962). Many of the models analysed throughout the thesis can be viewed as finite state, finite action MDPs. In these cases, the model assumptions ensure that the average reward per unit time exists for the policies considered. When maximising $R^u(\mathbf{k}, \alpha)$ over all admissible policies $u \in \mathcal{U}$, it is clear from (2.32) that for small α, prime interest is in the first term of the corresponding expansion. We say policy u is **average reward optimal** if

$$R_1^u(\mathbf{k}) = \sup_{v \in \mathcal{U}} R_1^v(\mathbf{k}). \tag{2.33}$$

Under given conditions, $R_1^v(\mathbf{k})$ is independent of $\mathbf{k}$. The average reward criterion can be strengthened by consideration of the second term in the expansion (2.32). Following the results of Blackwell (1962), Veinott (1966) introduced the stronger average overtaking optimality criterion in the MDP framework. Consider the class of policies $\mathcal{U}'$ satisfying (2.33); any policy $u' \in \mathcal{U}'$ is **average overtaking optimal** if

$$R_2^{u'}(\mathbf{k}) = \sup_{v \in \mathcal{U}'} R_2^v(\mathbf{k}). \tag{2.34}$$

Plainly a claim of average overtaking optimality implies a claim of average reward optimality.

This process can in principle be continued by examining the class of average overtaking optimal policies for policies achieving optimality in the $R_3(\mathbf{k})$ term and so on. Following the results of Blackwell (1962), if at any stage of this sequential process of analysis, the class of policies achieving

$$R_n^u(\mathbf{k}) = \sup_v R_n^v(\mathbf{k}), \quad n \geq 1,$$

contains a single policy, then there exists $\bar{\alpha} > 0$ such that for all $\alpha \in (0, \bar{\alpha}]$, this policy is optimal. This form of optimality is referred to as **Blackwell optimality**.

2.6 Numerical investigations

As explained at the beginning of Section 2.5, a key feature of the thesis is the derivation of bounds on the degree of reward suboptimality of index based controls. In Chapters 3 and 4, for the models and problem instances considered, in order to investigate the tightness of the bounds more widely, computational experiments are presented. The tightest suboptimality bounds obtained from the primal dual theory are compared numerically with actual reward suboptimalities computed using the Value Iteration Algorithm of dynamic programming. An outline of the Value Iteration Algorithm is given below. These comparisons assess the degree of reward suboptimality of index based heuristics developed from the theory and offer insights concerning the degree of conservatism in the theoretical results. By determining the performance of the heuristics over varying discount rates, the behaviour of the bounds as $\alpha \to 0$ is investigated.

The Value Iteration Algorithm

As mentioned in Section 1.2, dynamic programming formulations can, in principal at least, yield numerical solutions for relatively small discounted Markov decision problems. Such methods rely on making use of the evolutionary characteristics to formulate an optimality equation. Consider a discrete time Markov decision process observed at time $t \in \mathbb{N}$ to be in one of a countable set of states $\mathbf{X} = \{\mathbf{1}, \mathbf{2}, \ldots, |\mathbf{X}|\}$, where $\mathbf{X}$ may be, for example, the number of each job class present in the system or the current

states of a collection of bandits. Having observed the state of the process, at each decision epoch, an action must be chosen from the set of possible actions. The action sets $A(\mathbf{x})$, $\mathbf{x} \in \mathbf{X}$ are assumed to be finite. If action a is chosen when the process is in state $\mathbf{x}$ at time t a reward of $r(\mathbf{x}, a) \int_t^{t+1} e^{-\alpha s} ds$ is earned and with probability $P_{\mathbf{x}\mathbf{x}'}(a)$ the process will be in state $\mathbf{x}' \in \mathbf{X}$ at time $t+1$. The Value Iteration Algorithm of dynamic programming is a common method for solving problems of this type.

Let $R_t^{\text{opt}}(\mathbf{x})$ be the optimal expected discounted reward earned, starting from state $\mathbf{x}$ at time zero for a t period problem. The optimality equation is written

$$R_t^{\text{opt}}(\mathbf{x}) = \max_{a \in A(\mathbf{x})} \left\{ r(\mathbf{x}, a) \int_0^1 e^{-\alpha s} ds + e^{-\alpha} \sum_{\mathbf{x}' \in \mathbf{X}} P_{\mathbf{x}\mathbf{x}'}(a) R_{t-1}^{\text{opt}}(\mathbf{x}') \right\}. \tag{2.35}$$

The Value Iteration Algorithm computes recursively for $t = 1, 2, \ldots$ with

$$R_0^{\text{opt}}(\mathbf{x}) = \max_{a \in A(\mathbf{x})} \left\{ r(\mathbf{x}, a) \int_0^1 e^{-\alpha s} ds \right\} \qquad \mathbf{x} \in \mathbf{X}.$$

Clearly controls maximising the expression in (2.35) are optimal for the process when truncated at time t. Standard results from dynamic programming, see for example Ross (1970), show that

$$R_t^{\text{opt}}(\mathbf{x}) \to R^{\text{opt}}(\mathbf{x}) \text{ as } t \to \infty, \tag{2.36}$$

where $R^{\text{opt}}(\mathbf{x})$ is the infinite horizon optimal reward. For the purposes of calculating the optimal expected reward for a particular problem, the algorithm is stopped when it converges to within a pre-specified tolerance which we define as follows; let $T \in \mathbb{N}$ be the number of iterations required until convergence. We define the tolerance $\epsilon \in \mathbb{R}^+$ such that

$$R^{\text{opt}}(\mathbf{x}) - R_T^{\text{opt}}(\mathbf{x}) \leq \epsilon. \tag{2.37}$$

We infer from (2.35) that

$$R^{\text{opt}}(\mathbf{x}) - R_T^{\text{opt}}(\mathbf{x}) \leq \max_{\mathbf{x} \in \mathbf{X} a \in A(\mathbf{x})} \{ r(\mathbf{x}, a) \} \int_T^{\infty} e^{-\alpha s} ds. \tag{2.38}$$

It follows from (2.37) and (2.38) that for tolerance ϵ it is sufficient that

$$T \geq \left(\frac{1}{\alpha} \right) \ln \left[\frac{\max_{\mathbf{x} \in \mathbf{X} a \in A(\mathbf{x})} \{ r(\mathbf{x}, a) \}}{\alpha \epsilon} \right].$$

Clearly, since $\ln(s)$ is increasing in s, as the discount rate $\alpha \to 0$ the number of iterations of the algorithm required until convergence, $T \to \infty$ rapidly, which is a major

disadvantage computationally. General computational problems also occur with the Value Iteration Algorithm when the state space $\mathbf{X}$ is large.

The Value Iteration Algorithm can be used to compute optimal rewards and the rewards from the index based controls, such as those discussed in later chapters. Hence corresponding suboptimality bounds follow. The computation of rewards earned when implementing index based heuristics uses a similar approach to that outlined above.

Plainly in order to evaluate the rewards earned when following index based heuristics for specific problems, the Gittins indices must be computed explicitly. This can be achieved via AG outlined in Section 1.5.3, in which case access to the matrix $\mathbf{A}$ is required. For details of the methods used to compute $\mathbf{A}$ for the branching bandit and multi-armed bandit problems, see Bertsimas and Niño-Mora (1996).

The computer programs used to conduct the numerical studies are written in the programming language Fortran 95 and are given in Appendices A-I. To obtain a variety of profiles, the programs generate random bandit processes with random transition matrices and rewards, for supplied values of the discount rate α and other variables relevant to the specific problem. For each problem the suboptimality for a given policy is determined using the Value Iteration Algorithm. A number of processes are randomly generated and the summary statistics from these simulations are presented. The details of specific programs are given in the relevant sections throughout the thesis.

Chapter 3

Multi-armed Bandit Problems on Parallel Machines with varying speeds

3.1 Introduction

This chapter considers the discounted multi-armed bandit problem on a collection of machines, working in parallel but possibly at different speeds. Weiss (1982) considers the optimal scheduling of a batch of jobs on such a collection of machines. In his work, each job has a processing requirement which is exponentially distributed but the associated cost rates are quite general. He gives conditions under which simple priority policies which order the jobs according to their mean processing times are optimal. More recent work focuses on the issue of whether to place a job on a machine now, or wait for a faster machine to become available. A survey of contributions along these lines is given by Weiss (1995).

The model we use is described in Section 3.2, and tools of analysis are developed in Section 3.3. We shall consider both general (non-anticipative, non-idling) policies and also a restriction to a class of policies comparable to those analysed in Section 2.4. These block allocation policies make a single irrevocable decision at time zero concerning which machine will have exclusive rights to the processing of each bandit. Section 3.4 focuses on such policies and describes how Blackwell optimality may be achieved under given conditions along with weaker forms of asymptotic optimality more generally. Roughly

speaking, the optimal policy matches the bandits with the largest guaranteed reward rates with the machines operating at the greatest speed. The block allocation policy identified is shown to be -1-discount optimal and average reward optimal in the class of general policies in Section 3.5. Section 3.6 discusses the performance of the Gittins index policy when implemented in the parallel machine environment with machines operating at different speeds, offering a comparison with the performance of the block allocation policy identified in Section 3.4. In each of the Sections 3.4 - 3.6, we use the Value Iteration Algorithm to obtain the associated reward suboptimalities computationally allowing insights concerning the degree of conservatism in the theoretical bounds.

3.2 The Model

There are B bandits or projects which are available for processing by a collection of M machines. We suppose that $B > M$. Each bandit b evolves under processing within finite state space E_b. Write $E = \bigcup_b E_b$ for the disjoint union of the individual bandit state spaces. Bandit b's evolution under processing is determined by the irreducible one-step transition matrix $\mathbf{P}^b$, while the rewards it earns are given by the positive-valued reward vector $\mathbf{r}^b = \{r_i,\ i \in E_b\}$. Each machine can process all B bandits but different machines work at different speeds. Machine m works at speed $s_m > 0$, meaning that with machine m is associated the sequence of decision epochs $\{t/s_m,\ t \in \mathbb{N}\}$. During each associated time interval $[t/s_m, (t+1)/s_m)$, machine m effects a single transition within any bandit submitted to it for processing. Specifically, should bandit b be submitted to machine m at epoch t/s_m when in state $i \in E_b$ then with probability P_{ij}^b it will be in state $j \in E_b$ at time $(t+1)/s_m$. Recall from Section 1.3, under traditional formulations of multi-armed bandit problems, the reward earned from this transition is $r_i e^{-\alpha t/s_m}$ where $\alpha > 0$ is a discount rate. To simplify computations however, observe that

$$r_i e^{-\alpha t/s_m} = r_i \left(\int_{t/s_m}^{(t+1)/s_m} e^{-\alpha s} ds \right) / \left(\int_0^{1/s_m} e^{-\alpha s} ds \right).$$

Hence the total expected return from the processing on machine m under control u may be written

$$\sum_{i \in E} \left\{ r_i / \int_0^{1/s_m} e^{-\alpha s} ds \right\} x_{im}^u(\alpha), \tag{3.1}$$

where

$$x_{im}^{u}(\alpha) = E_u \left\{ \int_0^\infty \chi_{im}(s) e^{-\alpha s} ds \right\},$$

with

$$\chi_{im}(s) = \begin{cases} 1, & \text{if at time } s \text{ a bandit in state } i \text{ is being processed on machine } m, \\ 0, & \text{otherwise,} \quad i \in E,\ 1 \leq m \leq M. \end{cases}$$

It follows that the reward earned by policy u may be written

$$R^u = \sum_{m=1}^{M} \sum_{i \in E} \left\{ r_i / \int_0^{1/s_m} e^{-\alpha s} ds \right\} x_{im}^{u}(\alpha). \tag{3.2}$$

Note that, while R^u depends upon the system's initial state, this dependence will be suppressed in the notation. The goal of optimisation is to choose a policy u from the admissible class to maximise R^u.

We shall consider two classes of admissible policies for allocating bandits to machines:

(a) **Block allocation policies**.

Before processing begins, a block allocation policy makes a once-for-all allocation of bandits to machines (from the M^B available). Each machine m then chooses a single bandit from its allocated collection, B_m say, to process at each decision epoch t/s_m, $t \in \mathbb{N}$, according to a non-anticipative and non-idling policy.

(b) **General policies**.

A general policy u will make allocation decisions at each $\tau \in \bigcup_{m=1}^{M} \{t/s_m,\ t \in \mathbb{N}\}$. Any bandit which at τ has **not** been allocated to a machine whose current processing interval goes beyond τ may be allocated at τ to any machine for which τ is a decision epoch. Any such allocation must be made in a non-anticipative way. Hence the allocation decision at τ may take account of all allocations made and transitions observed prior to τ. Additionally, all M machines should always be kept busy. Plainly, the class of general policies contains the class of block allocation policies.

In what follows, we shall use $\mathcal{U}$ to denote the class of admissible policies under consideration.

3.3 Tools of analysis

Fix $i \in E_b$ and subset $S \subseteq E$ with $i \in S$. Consider a set up in which only bandit b in state i is present at time 0. Processing is *standard*, namely provided by a single machine operating at unit speed. Under such processing, bandit b evolves as a Markov chain making transitions at time $t \in \mathbb{Z}^+$ according to one-step transition matrix $\mathbf{P}^b$. As in Section 2.3 we write $T_i^{S^c}$ for the first time at or after time 1 at which bandit b enters S. Recall that from the assumed irreducibility (and hence positive recurrence) of the Markov chain, it follows that that all positive moments of $T_i^{S^c}$ are finite. As in (2.6) we define the vector $\mathbf{A}^S(\alpha)$ as

$$A_i^S(\alpha) = \begin{cases} \left\{1 - E\left(e^{-\alpha T_i^{S^c}}\right)\right\} / \left(1 - e^{-\alpha}\right), & i \in S, \\ 0, & i \notin S. \end{cases} \tag{3.3}$$

Observe that $A_i^E(\alpha) = 1$, $i \in E$, $\alpha > 0$. The so-called adaptive greedy algorithm AG, as in Section 1.5.3, has inputs given by the collection $\mathbf{A}(\alpha) = \{\mathbf{A}^S(\alpha),\ S \subseteq E\}$ together with the reward vector $\mathbf{r} = \{r_j,\ j \in E\}$. The outputs from the algorithm are a set of non-negative real numbers $G_j(\alpha)$, $j \in E$, which we shall call *standard Gittins indices*. Note we include the dependence on α in the notation. Key results by Katehakis and Veinott (1987) characterise the quantities $G_j(\alpha)/(1 - e^{-\alpha})$, $j \in E$, as values of a special class of Markov Decision Chains (MDCs) called *restart problems*. This, together with classical results regarding series expansions for the value functions of MDCs (recall (2.32)), implies that we may write

$$G_j(\alpha) = G_j + \alpha g_j + O(\alpha^2),\ j \in E, \tag{3.4}$$

where $G_j \equiv \lim_{\alpha \to 0} G_j(\alpha)$.

Consider now the processing on machine m, at speed s_m. Since transitions occur at times t/s_m, $t \in \mathbb{Z}^+$, then the form of the vector $\mathbf{A}_m^S(\alpha)$ we require for machine m is given by

$$\begin{aligned} A_{i,m}^S(\alpha) &= \begin{cases} \left\{1 - E\left(e^{-\alpha T_i^{S^c}/s_m}\right)\right\} / \left(1 - e^{-\alpha/s_m}\right), & i \in S, \\ 0, & i \notin S \end{cases} \\ &= \mathbf{A}^S\left(\frac{\alpha}{s_m}\right),\ 1 \le m \le M. \end{aligned} \tag{3.5}$$

Further, the Gittins indices required for the processing on machine m are $G_j(\alpha/s_m)$, $j \in E$, $1 \le m \le M$.

We now make an assumption which will considerably simplify the theoretical development. The effects of relaxing this assumption are discussed in Section 3.7.

Assumption 1

There are no distinct members j, k of E for which $G_j = G_k,\ g_j = g_k$.

Consider now the unique numbering of E for which

$$j > k \Rightarrow \text{ either } G_j > G_k \text{ or } G_j = G_k,\ g_j > g_k. \tag{3.6}$$

From (3.4) and (3.6) it follows that there must exist α^* such that for all $\alpha \in (0, \alpha^*)$ the ordering $|E| \to |E|-1 \to \ldots \to 2 \to 1$ is a Gittins index ordering on E for all machines m, i.e.

$$G_{|E|}\left(\frac{\alpha}{s_m}\right) \geq G_{|E|-1}\left(\frac{\alpha}{s_m}\right) \geq \ldots \geq G_2\left(\frac{\alpha}{s_m}\right) \geq G_1\left(\frac{\alpha}{s_m}\right),$$
$$\alpha \in (0, \alpha^*),\ 1 \leq m \leq M. \tag{3.7}$$

We shall use this numbering of members of E and suppose henceforth that $\alpha \in (0, \alpha^*)$. As in earlier chapters, we write $S_j = \{j, j-1, \ldots, 1\}$ for the subset of E of cardinality j with lowest identifiers. Following (1.31), from the structure of AG the rewards $\{r_i, i \in E\}$ may be expressed as

$$r_i = G_{|E|}\left(\frac{\alpha}{s_m}\right) A_i^E\left(\frac{\alpha}{s_m}\right) - \sum_{j=i}^{|E|-1} \left\{ G_{j+1}\left(\frac{\alpha}{s_m}\right) - G_j\left(\frac{\alpha}{s_m}\right) \right\} A_i^{S_j}\left(\frac{\alpha}{s_m}\right),$$
$$i \in E,\ 1 \leq m \leq M. \tag{3.8}$$

Now write

$$G_j^m(\alpha) = G_j\left(\frac{\alpha}{s_m}\right) \Big/ \int_0^{1/s_m} e^{-\alpha s} ds \tag{3.9}$$

$$= \frac{G_j + (\alpha/s_m) g_j + O(\alpha^2)}{(1/\alpha)(1 - e^{-\alpha/s_m})}$$

$$= \frac{G_j + (\alpha/s_m) g_j + O(\alpha^2)}{1/s_m - \alpha/(2s_m^2) + O(\alpha^2)}$$

$$= \frac{s_m G_j + \alpha g_j + O(\alpha^2)}{1 - \{\alpha/(2s_m) + O(\alpha^2)\}}$$

$$= \Big(s_m G_j + \alpha g_j\Big)\Big(1 + \alpha/(2s_m)\Big) + O(\alpha^2)$$

$$= s_m G_j + \alpha(g_j + \frac{1}{2} G_j) + O(\alpha^2), \quad j \in E,\ 1 \le m \le M. \tag{3.10}$$

Observe from (3.7) that we must have

$$G_{|E|}^m(\alpha) \ge G_{|E|-1}^m(\alpha) \ge \ldots \ge G_2^m(\alpha) \ge G_1^m(\alpha),\ \alpha \in (0, \alpha^*),\ 1 \le m \le M.$$

Utilising (3.8) and (3.9) in (3.2) we obtain

$$R^u = \sum_{m=1}^{M} G_{|E|}^m(\alpha) \sum_{i \in E} A_i^E\left(\frac{\alpha}{s_m}\right) x_{im}^u(\alpha)$$
$$- \sum_{m=1}^{M} \sum_{j=1}^{|E|-1} \{G_{j+1}^m(\alpha) - G_j^m(\alpha)\} \sum_{i \in S_j} A_i^{S_j}\left(\frac{\alpha}{s_m}\right) x_{im}^u(\alpha)$$

$$= \frac{\sum_{m=1}^{M} G_{|E|}^m(\alpha)}{\alpha} - \sum_{m=1}^{M} \sum_{j=1}^{|E|-1} \{G_{j+1}^m(\alpha) - G_j^m(\alpha)\} A^{m,u}(S_j, \alpha), \tag{3.11}$$

where in (3.11) we use the notational shorthand

$$A^{m,u}(S, \alpha) = \sum_{i \in S} A_i^S\left(\frac{\alpha}{s_m}\right) x_{im}^u(\alpha)$$

and the fact that for all controls u

$$A^{m,u}(E, \alpha) = \sum_{i \in E} x_{im}^u(\alpha) = \int_0^\infty e^{-\alpha s} ds = 1/\alpha, \quad 1 \le m \le M.$$

We now use (3.10) and the fact that $A^{m,u}(S_j, \alpha) \le O(1/\alpha)$ to simplify the expression in (3.11) for R^u. To further simplify notation, we write $\delta_j = g_j + \frac{1}{2} G_j,\ j \in E$.

Theorem 1

For all policies $u \in \mathcal{U}$,

$$\begin{aligned}\frac{\sum_{m=1}^{M} G_{|E|}^{m}(\alpha)}{\alpha} - R^{u} &= \sum_{m=1}^{M}\sum_{j=1}^{|E|-1}\left\{G_{j+1}^{m}(\alpha) - G_{j}^{m}(\alpha)\right\} A^{m,u}(S_j,\alpha) \\ &= \sum_{j=1}^{|E|-1}(G_{j+1}-G_j)\sum_{m=1}^{M} s_m A^{m,u}(S_j,\alpha) \\ &\quad +\alpha\sum_{j=1}^{|E|-1}(\delta_{j+1}-\delta_j)\sum_{m=1}^{M} A^{m,u}(S_j,\alpha) + O(\alpha).\end{aligned}$$

The following is an immediate consequence of Theorem 1.

Corollary 1

For all policies $u \in \mathcal{U}$,

$$\begin{aligned}R^{\mathrm{opt}} - R^{u} &= \sum_{j=1}^{|E|-1}(G_{j+1}-G_j)\left\{\sum_{m=1}^{M} s_m A^{m,u}(S_j,\alpha) - \sum_{m=1}^{M} s_m A^{m,\mathrm{opt}}(S_j,\alpha)\right\} \\ &\quad +\alpha\sum_{j=1}^{|E|-1}(\delta_{j+1}-\delta_j)\left\{\sum_{m=1}^{M} A^{m,u}(S_j,\alpha) - \sum_{m=1}^{M} A^{m,\mathrm{opt}}(S_j,\alpha)\right\} + O(\alpha) \qquad (3.12) \\ &\leq \sum_{j=1}^{|E|-1}(G_{j+1}-G_j)\left\{\sum_{m=1}^{M} s_m A^{m,u}(S_j,\alpha) - A(S_j,\alpha)\right\} \\ &\quad +\alpha\sum_{j=1}^{|E|-1}(\delta_{j+1}-\delta_j)\left\{\sum_{m=1}^{M} A^{m,u}(S_j,\alpha) - \sum_{m=1}^{M} A^{m,\mathrm{opt}}(S_j,\alpha)\right\} + O(\alpha), \qquad (3.13)\end{aligned}$$

where

$$A(S,\alpha) = \inf_{\nu\in\mathcal{U}}\sum_{m=1}^{M} s_m A^{m,\nu}(S,\alpha),\ S \subseteq E.$$

Proof

Inequality (3.12) is an immediate consequence of Theorem 1. Inequality (3.13) follows from (3.12) and the facts that, from (3.6),

$$G_{j+1} - G_j \geq 0, \quad 1 \leq j \leq |E| - 1,$$

and

$$\sum_{m=1}^{M} s_m A^{m,\mathrm{opt}}(S_j, \alpha) \geq A(S_j, \alpha), \quad 1 \leq j \leq |E| - 1.$$

This concludes the proof.

Note that henceforth the machines are numbered such that $s_1 \geq s_2 \geq \ldots \geq s_M$.

3.4 Optimal block allocation policies

We begin by considering the class of **block allocation policies** which partition the set of bandits according to a disjoint union

$$\bigcup_{m=1}^{M} B_m. \tag{3.14}$$

Use $\mathbf{B}$ to denote the partition in (3.14).

According to the block allocation policy corresponding to $\mathbf{B}$, the bandits in B_m are processed exclusively on machine m from time 0 onwards, $1 \leq m \leq M$. By the classical result of Gittins and Jones (1974), the total expected reward from this partition will be optimised by using a Gittins index ordering $|E| \to |E| - 1 \to \ldots \to 2 \to 1$ to schedule the bandits on each machine. We use the notational shorthand $u_G(\mathbf{B})$ for this policy. The goal of analysis is to optimise $R^{u_G(\mathbf{B})}$ over all choices of partition $\mathbf{B}$. Note that this class of policies is comparable to those analysed in Section 2.4.

Recall the numbering of E in (3.6). As in Chapter 2, we write

$$j(b) = \min\{j;\ j \in E_b\},\ 1 \leq b \leq B,$$

for the minimum index state for bandit b. We also write

$$j(B_m) = \max\{j(b);\ b \in B_m\},\ 1 \leq m \leq M,$$

for the maximal such state in the collection B_m. The significance of $j(B_m)$ is that it is the member of B_m with smallest identifier which will ever be processed on machine m, $1 \leq m \leq M$, under policy $u_G(\mathbf{B})$. Further, considering the interpretation of (1.6) in Section 1.3, $G_{j(b)}$ can be thought of as a (minimal limiting) guaranteed reward rate for bandit b. Now fix $j \geq j(B_m)$ and use $T_{B_m}^{S_j^c}$ to denote the first occasion at or after

time 0 at which all members of B_m are in states in S_j when policy $u_G(\mathbf{B})$ is operating and standard (i.e. unit speed) processing is assumed. We write

$$T_{B_m}^{S_j^c} = \sum_{b \in B_m} T_b^{S_j^c} \tag{3.15}$$

to represent the decomposition of $T_{B_m}^{S_j^c}$ which reveals the contributions from the individual bandits within B_m.

Lemma 2

$$A^{m,u_G(\mathbf{B})}(S_j, \alpha) = \begin{cases} \frac{1}{\alpha} E\{\exp(-\alpha T_{B_m}^{S_j^c}/s_m)\}, & j \geq j(B_m), \\ 0, & j < j(B_m). \end{cases} \tag{3.16}$$

Proof

By the structure of $u_G(\mathbf{B})$, no job type $j < j(B_m)$ will ever be processed by machine m, $1 \leq m \leq M$, since a job of type $j \geq j(B_m)$ will always be preferred. Hence

$$A^{m,u_G(\mathbf{B})}(S_j, \alpha) = 0 \qquad j < j(B_m).$$

We denote by $\tau_{i,l,m}$, the time of the l^{th} occasion upon which control $u_G(\mathbf{B})$ processes bandit $b \in B_m$ in state $i \in S_j$. The associated collection of independent and identically distributed random variables $\{T_{i,l,m}^{S_j^c}, l \in \mathbb{Z}^+\}$ share the distribution of $T_i^{S_j^c}$ defined at the beginning of Section 3.3. Under $u_G(\mathbf{B})$, for machine m, the first contribution to $A^{m,u_G(\mathbf{B})}(S_j, \alpha)$ will occur at time $T_{B_m}^{S_j^c}/s_m$, the first occasion at which a job in S_j is chosen. Following this epoch, excursions to S_j^c will alternate with allocations to S_j. We write

$$E_{B_m} = \bigcup_{b \in B_m} E_b.$$

It follows from the definitions of the quantities involved, that

$$\begin{aligned} A^{m,u_G(\mathbf{B})}(S_j, \alpha) &= \sum_{i \in S_j \cap E_{B_m}} E_{u_G(\mathbf{B})}\Big(\sum_{l=1}^{\infty} \int_{\tau_{i,l,m}}^{\tau_{i,l,m} + T_{i,l,m}^{S_j^c}/s_m} e^{-\alpha t} dt \,\Big|\, k(m) \Big) \\ &= \frac{1}{\alpha} E\Big(\exp[-\alpha T_{B_m}^{S_j^c}/s_m] \Big), \qquad j \geq j(B_m). \end{aligned}$$

This concludes the proof.

We now substitute from (3.16) into the expression for $R^{u_G(\mathbf{B})}$ given in Theorem 1 to obtain the following:

Corollary 2

$$R^{u_G(\mathbf{B})} = \frac{\sum_{m=1}^{M} s_m G_{j(B_m)}}{\alpha} + \sum_{m=1}^{M} \sum_{j=j(B_m)}^{|E|-1} (G_{j+1} - G_j) E(T_{B_m}^{S_j^c}) + \sum_{m=1}^{M} \delta_{j(B_m)} + O(\alpha). \quad (3.17)$$

Proof

Substituting from (3.10) into Theorem 1 and utilising Lemma 2 and the existence of all positive moments of $T_{B_m}^{S_j^c}$ we obtain

$$\begin{aligned}
R^{u_G(\mathbf{B})} &= \frac{\sum_{m=1}^{M} G_{|E|}^m(\alpha)}{\alpha} - \sum_{m=1}^{M} \sum_{j=1}^{|E|-1} \left\{ G_{j+1}^m(\alpha) - G_j^m(\alpha) \right\} A^{m,u_G(\mathbf{B})}(S_j, \alpha) \\
&= \frac{\sum_{m=1}^{M} s_m G_{|E|}}{\alpha} - \sum_{m=1}^{M} \sum_{j=j(B_m)}^{|E|-1} (G_{j+1} - G_j)\, s_m A^{m,u_G(\mathbf{B})}(S_j, \alpha) \\
&\quad + \sum_{m=1}^{M} \delta_{|E|} - \sum_{m=1}^{M} \sum_{j=j(B_m)}^{|E|-1} (\delta_{j+1} - \delta_j)\, \alpha A^{m,u_G(\mathbf{B})}(S_j, \alpha) + O(\alpha) \\
&= \frac{\sum_{m=1}^{M} s_m G_{|E|}}{\alpha} - \sum_{m=1}^{M} \sum_{j=j(B_m)}^{|E|-1} (G_{j+1} - G_j)\, s_m E\Big[(1/\alpha) - (T_{B_m}^{S_j^c}/s_m) + O(\alpha) \Big] \\
&\quad + \sum_{m=1}^{M} \delta_{|E|} - \sum_{m=1}^{M} \sum_{j=j(B_m)}^{|E|-1} (\delta_{j+1} - \delta_j)\, \alpha E\Big[(1/\alpha) - (T_{B_m}^{S_j^c}/s_m) + O(\alpha) \Big] + O(\alpha) \\
&= \frac{\sum_{m=1}^{M} s_m G_{|E|}}{\alpha} - \sum_{m=1}^{M} \sum_{j=j(B_m)}^{|E|-1} \frac{s_m (G_{j+1} - G_j)}{\alpha} + \sum_{m=1}^{M} \sum_{j=j(B_m)}^{|E|-1} (G_{j+1} - G_j)\, E(T_{B_m}^{S_j^c}) \\
&\quad + \sum_{m=1}^{M} \delta_{|E|} - \sum_{m=1}^{M} \sum_{j=j(B_m)}^{|E|-1} (\delta_{j+1} - \delta_j) + O(\alpha) \\
&= \frac{\sum_{m=1}^{M} s_m G_{j(B_m)}}{\alpha} + \sum_{m=1}^{M} \sum_{j=j(B_m)}^{|E|-1} (G_{j+1} - G_j) E(T_{B_m}^{S_j^c}) + \sum_{m=1}^{M} \delta_{j(B_m)} + O(\alpha),
\end{aligned}$$

as required.

From (3.17) it is plain that, for sufficiently small α, the first step in optimising $R^{u_G(\mathbf{B})}$ is to consider partitions $\mathbf{B}$ which maximise the quantity $\sum_{m=1}^{M} s_m G_{j(B_m)}$. We focus our analysis initially on a special case where it is clear how this is done. To simplify our account, we first require that the bandits are numbered such that

$$j(1) > j(2) > \ldots > j(B).$$

Theorem 2

If the following conditions hold:

$$G_{j(1)} > G_{j(2)} > \ldots > G_{j(M)} > G_{j(M+1)} \geq G_{j(M+2)} \geq \ldots \geq G_{j(B)},$$

and

$$s_1 > s_2 > \ldots > s_M,$$

then there exists $\bar{\alpha} > 0$ such that for all $\alpha \in (0, \bar{\alpha})$ the policy $u_G(\mathbf{B}^*)$ determined by the partition

$$B_m^* = \begin{cases} \{m\}, & 1 \leq m \leq M-1, \\ \{M, M+1, \ldots, B\}, & m = M, \end{cases}$$

is optimal in the class of block allocation policies.

Proof

It is clear that under the conditions in the statement of the result,

$$\sum_{m=1}^{M} s_m G_{j(B_m)} \leq \sum_{m=1}^{M} s_m G_{j(m)}, \tag{3.18}$$

with strict inequality in (3.18) for any partition which fails to have $m \in B_m$, $1 \leq m \leq M$. Hence, from Corollary 2, for sufficiently small α, $R^{u_G(\mathbf{B})}$ must be optimised by a partition of the form

$$B_m = \{m\} \cup \beta_m, \ 1 \leq m \leq M, \tag{3.19}$$

where

$$\bigcup_{m=1}^{M} \beta_m = \{M+1, M+2, \ldots, B\}.$$

We now utilise (3.15) within Corollary 2 and substitute from (3.19) to obtain, for any such partition, that

$$\begin{aligned} R^{u_G(\mathbf{B})} &= \frac{\sum_{m=1}^{M} s_m G_{j(B_m)}}{\alpha} + \sum_{m=1}^{M} \sum_{j=j(B_m)}^{|E|-1} (G_{j+1} - G_j) \Big\{ \sum_{b \in B_m} E(T_b^{S_j^c}) \Big\} + \sum_{m=1}^{M} \delta_{j(B_m)} + O(\alpha) \\ &= \frac{\sum_{m=1}^{M} s_m G_{j(m)}}{\alpha} + \sum_{m=1}^{M} \sum_{j=j(m)}^{|E|-1} (G_{j+1} - G_j) E(T_m^{S_j^c}) \\ &\quad + \sum_{m=1}^{M} \delta_{j(m)} + \sum_{m=1}^{M} \sum_{j=j(m)}^{|E|-1} (G_{j+1} - G_j) \Big\{ \sum_{b \in \beta_m} E(T_b^{S_j^c}) \Big\} + O(\alpha). \end{aligned} \tag{3.20}$$

Examining (3.20), we note that the choice of β_m, $1 \leq m \leq M$, does not affect any of the first three terms on the right-hand side of the equation. Focussing on the fourth term, it follows that the contribution from bandit $b \in \beta_m$ to this sum is

$$\sum_{j=j(m)}^{|E|-1} (G_{j+1} - G_j) E(T_b^{S_j^c}) \leq \sum_{j=j(M)}^{|E|-1} (G_{j+1} - G_j) E(T_b^{S_j^c}), \tag{3.21}$$

since $j(m) \geq j(M)$, $1 \leq m \leq M$, and all the summands in (3.21) are non-negative. From (3.21) it is straightforward to deduce that the fourth term on the right-hand side of (3.20) is maximised by the choice

$$\beta_m^* = \begin{cases} \emptyset, & 1 \leq m \leq M-1, \\ \{M+1, M+2, \ldots, B\}, & m = M. \end{cases} \tag{3.22}$$

In fact if some bandit $b \in \{M+1, M+2, \ldots, B\}$ is in a state within $S_{j(M)}^c$ then the inequality in (3.21) is strict for that b and for any $m \leq M-1$. Moreover, if at time $t = 0$, bandit $b \in \{M+1, M+2, \ldots, B\}$ is in a state within $S_{j(M)}$ then it can be excluded from consideration since it will never be processed under policy $u_G(\mathbf{B})$ for any $\mathbf{B}$ of the form in (3.19). Hence the choice in (3.22) uniquely maximises the fourth term in (3.20), modulo the distribution of bandits which do not contribute to the expected rewards. The result follows easily.

Comment

Observe from the proof of Theorem 2 that the maximisation of the $O(1/\alpha)$ term in $R^{u_G(\mathbf{B})}$ forces the conclusion $m \in B_m$, $1 \leq m \leq M$. The destination of the remaining bandits is settled upon consideration of the $O(1)$ term. The form of optimality claimed for $u_G(\mathbf{B}^*)$ is **Blackwell optimality**, outlined in Section 2.5. Note that $u_G(\mathbf{B}^*)$ allocates bandits in decreasing order of guaranteed reward rate to machines in decreasing order of speed.

Theorem 3

Under general conditions, the partition

$$B_m^* = \begin{cases} \{m\}, & 1 \leq m \leq M-1, \\ \{M, M+1, \ldots, B\}, & m = M, \end{cases}$$

satisfies

$$R^{\text{opt}} - R^{u_G(\mathbf{B}^*)} \leq O(\alpha),$$

where the optimum is taken over the class of block allocation policies.

Proof

We define $\tilde{m}$ as follows:

$$\tilde{m} = \min\{m;\ G_{j(m)} = G_{j(M)}\}. \tag{3.23}$$

Recall the comments following Corollary 2. When optimising $R^{u_G(\mathbf{B})}$ for sufficiently small α, we firstly consider partitions $\mathbf{B}$ which maximise the quantity $\sum_{m=1}^{M} s_m G_{j(B_m)}$. Hence we may assume that for small enough α, any optimal partition $\mathbf{B}$ must (modulo permuting bandits among machines of equal speed) distribute the bandits$\{1, 2, \ldots, \tilde{m}-1\}$ among the machines $\{1, 2, \ldots, \tilde{m}-1\}$, one to each machine, in a way which guarantees that

$$\sum_{m=1}^{\tilde{m}-1} s_m G_{j(B_m)} = \sum_{m=1}^{\tilde{m}-1} s_m G_{j(m)}.$$

It is clear that we must also have

$$G_{j(B_m)} = G_{j(M)}, \quad \tilde{m} \leq m \leq M, \tag{3.24}$$

for any optimal partition. Restricting to such partitions, from Corollary 2 we may write

$$\begin{aligned} R^{u_G(\mathbf{B})} \;=\; & \left\{ \sum_{m=1}^{\tilde{m}-1} s_m G_{j(m)} + \left(\sum_{m=\tilde{m}}^{M} s_m \right) G_{j(M)} \right\} / \alpha + \sum_{m=1}^{\tilde{m}-1} \sum_{j=j(B_m)}^{|E|-1} (G_{j+1} - G_j)\, E\left(T_{B_m}^{S_j^c}\right) \\ & + \sum_{m=\tilde{m}}^{M} \sum_{j=j(B_m)}^{|E|-1} (G_{j+1} - G_j)\, E\left(T_{B_m}^{S_j^c}\right) \\ & + \sum_{m=1}^{\tilde{m}-1} \delta_{j(B_m)} + \sum_{m=\tilde{m}}^{M} \delta_{j(B_m)} + O(\alpha). \end{aligned} \tag{3.25}$$

We observe that the second and fourth terms on the right hand side of (3.25) which involve the B_m, $1 \leq m \leq \tilde{m}-1$ are unaffected by a permutation of the bandit allocations among the machines $\{1, 2, \ldots, \tilde{m}-1\}$. Hence we may, without loss of generality, suppose in (3.25) that $m \in B_m$, $1 \leq m \leq \tilde{m}-1$. Focussing on the B_m, $\tilde{m} \leq m \leq M$, from (3.23) and (3.24), we conclude that

$$G_{j(\tilde{m})} = G_{j(B_m)} = G_{j(M)},\ \tilde{m} \leq m \leq M,$$

and hence that

$$G_{j+1} - G_j = 0, \quad \min\{j(B_m), j(m)\} \leq j \leq j(\tilde{m}) - 1, \quad \tilde{m} \leq m \leq M.$$

It follows from these observations and from (3.25) that the reward from any optimal partition may be written

$$\begin{aligned} R^{u_G(\mathbf{B})} &= \left\{ \sum_{m=1}^{\tilde{m}-1} s_m G_{j(m)} + \left(\sum_{m=\tilde{m}}^{M} s_m \right) G_{j(M)} \right\} / \alpha + \sum_{m=1}^{\tilde{m}-1} \sum_{j=j(m)}^{|E|-1} (G_{j+1} - G_j) \, E\left(T_{B_m}^{S_j^c} \right) \\ &\quad + \sum_{m=\tilde{m}}^{M} \sum_{j=j(m)}^{|E|-1} (G_{j+1} - G_j) \, E\left(T_{B_m}^{S_j^c} \right) + \sum_{m=1}^{\tilde{m}-1} \delta_{j(m)} + \sum_{m=\tilde{m}}^{M} \delta_{j(B_m)} + O(\alpha) \\ &= \left\{ \sum_{m=1}^{\tilde{m}-1} s_m G_{j(m)} + \left(\sum_{m=\tilde{m}}^{M} s_m \right) G_{j(M)} \right\} / \alpha + \sum_{m=1}^{M} \sum_{j=j(m)}^{|E|-1} (G_{j+1} - G_j) \, E\left(T_{B_m}^{S_j^c} \right) \\ &\quad + \sum_{m=1}^{\tilde{m}-1} \delta_{j(m)} + \sum_{m=\tilde{m}}^{M} \delta_{j(B_m)} + O(\alpha). \end{aligned} \tag{3.26}$$

We now utilise the argument in the proof of Theorem 2 around (3.21) to draw the conclusion that the second term on the right hand side of (3.26) is maximised over qualifying partitions by $\mathbf{B}^*$. Finally we observe that the ordering of E in (3.6) guarantees that

$$\sum_{m=1}^{\tilde{m}-1} \delta_{j(m)} + \sum_{m=\tilde{m}}^{M} \delta_{j(B_m)} \leq \sum_{m=1}^{M} \delta_{j(m)} = \sum_{m=1}^{M} \delta_{j(B_m^*)}.$$

We conclude from this analysis and from Corollary 2 that

$$\begin{aligned} R^{\text{opt}} &= \frac{\sum_{m=1}^{M} s_m G_{j(m)}}{\alpha} + \sum_{m=1}^{M} \sum_{j=j(m)}^{|E|-1} (G_{j+1} - G_j) E(T_{B_m^*}^{S_j^c}) + \sum_{m=1}^{M} \delta_{j(B_m^*)} + O(\alpha) \\ &= R^{u_G(\mathbf{B}^*)} + O(\alpha). \end{aligned}$$

The result follows.

Model assumptions guarantee that the expected rewards R^{opt} and $R^{u_G(\mathbf{B}^*)}$ are both $O(1/\alpha)$. Following Theorem 3 and the classical work of Blackwell (1962), Veinott (1966) and Denardo and Miller (1968), we take the further natural step in the analysis and explore the limit as $\alpha \to 0$. See Section 2.5. Various forms of asymptotic optimality are claimed for policy $u_G(\mathbf{B}^*)$ in Theorems 4 and 6. Since the policy $u_G(\mathbf{B}^*)$ is independent of α, it follows trivially from Theorem 3 that it is 0-discount optimal. Classical theory, together with the finiteness of the set of partitions $\mathbf{B}$, enables us to assert more.

Theorem 4 (Asymptotic optimality of block allocation policy $u_G(\mathbf{B}^*)$)

Under general conditions, $u_G(\mathbf{B}^*)$ is 0-discount optimal, average-overtaking optimal and average-reward optimal in the class of block allocation policies.

3.4.1 Numerical study

The performance of $u_G(\mathbf{B}^*)$ within the class of block allocation policies was investigated more widely via computational experiments. The computer program used to perform the study is given in Appendix A. We used the Value Iteration Algorithm to calculate the expected reward when following $u_G(\mathbf{B}^*)$ and also the optimal expected reward within the class of block allocation policies. This was straightforward since once the bandits have been partitioned, each machine can be viewed independently and hence the expected reward from each machine can be calculated as in Section 2.6 but with an adjustment in the discounting to account for the speed of the machine. The total expected reward in all cases is then the sum of the rewards from each individual machine.

The system studied is a four-armed bandit problem, each arm with four states, on two machines operating at varying speeds. For supplied values of the discount rate α and the speed of each machine, s_1 and s_2, the program generates a four-armed bandit problem with random reward vectors generated from a uniform $[l, u]$ for given l and u, a random starting state $\mathbf{k}$ and random irreducible transition matrices. Initially the entries in each transition matrix are generated from a uniform (0.1,0.9) then the matrices are normalised such that rows sum to 1. Since all entries of the resultant matrices are positive, the irreducibility assumed in Section 3.3 follows.

For each α from the set $\{0.5, 0.25, 0.1, 0.05, 0.025, 0.01, 0.005\}$ and set of speeds (s_1, s_2) from the set $\{(6,1), (5,2), (4.5,2.5), (4,3), (3.75,3.25), (3.625,3.375)\}$, 500 four-armed bandit problems were investigated where $r_i^b \sim U(2,5)$ and the tolerance ϵ as in (2.37) was 10^{-7}. The results of the study are given in Tables 3.1 - 3.6, each entry gives summary statistics for the 500 suboptimalities, $R^{\text{opt}} - R^{u_G(\mathbf{B}^*)}$, simulated. The summary statistics concerned are the **Maximum** and **Minimum** reward suboptimality of the 500 systems simulated, the **Mean** and standard deviation (**Std Dev**) of the 500 simulated suboptimalities and the **Median** of the positive reward suboptimalities. The results also include the percentage of the 500 reward suboptimalities that equal zero (**Percent**).

Alpha	Minimum	Median	Maximum	Mean	Std Dev	Percent
0.500	0.000000	0.100513	1.420304	0.114445	0.201518	33.60000
0.250	0.000000	0.077439	1.110234	0.072930	0.148596	50.00000
0.100	0.000000	0.054377	0.541196	0.022268	0.061542	73.80000
0.050	0.000000	0.037369	0.582096	0.011207	0.048533	85.40000
0.025	0.000000	0.023005	0.609888	0.005573	0.041123	90.80000
0.010	0.000000	0.013599	0.216960	0.001165	0.012798	96.80000
0.005	0.000000	0.004899	0.683718	0.001446	0.030573	98.80000

Table 3.1: Block allocation policies: Machine speeds 6 and 1

Alpha	Minimum	Median	Maximum	Mean	Std Dev	Percent
0.500	0.000000	0.071169	0.980946	0.059290	0.127924	52.00000
0.250	0.000000	0.043459	0.658720	0.029029	0.080374	67.40000
0.100	0.000000	0.027561	0.735803	0.007983	0.042840	86.40000
0.050	0.000000	0.017568	0.336667	0.003984	0.027255	92.00000
0.025	0.000000	0.006743	0.592945	0.003071	0.036565	94.20000
0.010	0.000000	0.007563	0.015150	0.000078	0.000903	99.00000
0.005	0.000000	0.130338	0.130338	0.000261	0.005823	99.80000

Table 3.2: Block allocation policies: Machine speeds 5 and 2

Alpha	Minimum	Median	Maximum	Mean	Std Dev	Percent
0.500	0.000000	0.062141	0.908553	0.048336	0.114323	56.60000
0.250	0.000000	0.035740	0.623175	0.021333	0.066921	72.80000
0.100	0.000000	0.023608	0.789139	0.006190	0.040514	88.00000
0.050	0.000000	0.014416	0.345472	0.003107	0.025497	93.80000
0.025	0.000000	0.006774	0.570148	0.002842	0.035989	96.00000
0.010	0.000000	0.005506	0.011825	0.000059	0.000693	99.00000
0.005	0.000000	0.000036	0.012152	0.000026	0.000544	99.00000

Table 3.3: Block allocation policies: Machine speeds 4.5 and 2.5

Alpha	Minimum	Median	Maximum	Mean	Std Dev	Percent
0.500	0.000000	0.056864	0.620557	0.035118	0.079244	62.40000
0.250	0.000000	0.042292	0.781757	0.021969	0.069743	73.20000
0.100	0.000000	0.031158	0.587503	0.010077	0.051704	87.60000
0.050	0.000000	0.005036	0.519563	0.001879	0.024120	94.00000
0.025	0.000000	0.002953	0.024913	0.000159	0.001610	97.60000
0.010	0.000000	0.002984	0.474316	0.001338	0.022600	98.20000
0.005	0.000000	0.000870	0.005073	0.000016	0.000240	99.00000

Table 3.4: Block allocation policies: Machine speeds 4 and 3

Alpha	Minimum	Median	Maximum	Mean	Std Dev	Percent
0.500	0.000000	0.053332	1.205313	0.042161	0.106130	61.00000
0.250	0.000000	0.038959	0.598741	0.018210	0.062636	76.80000
0.100	0.000000	0.022513	0.765492	0.011086	0.065858	87.80000
0.050	0.000000	0.008906	0.683600	0.003256	0.035559	94.00000
0.025	0.000000	0.016811	0.833528	0.003892	0.048927	97.40000
0.010	0.000000	0.000979	0.011682	0.000032	0.000535	98.80000
0.005	0.000000	0.001919	0.002677	0.000008	0.000130	99.60000

Table 3.5: Block allocation policies: Machine speeds 3.75 and 3.25

Alpha	Minimum	Median	Maximum	Mean	Std Dev	Percent
0.500	0.000000	0.059865	0.811859	0.037698	0.099986	65.00000
0.250	0.000000	0.034810	0.516165	0.015372	0.055414	79.20000
0.100	0.000000	0.018921	0.841021	0.004242	0.039539	91.20000
0.050	0.000000	0.008739	0.350454	0.002366	0.023038	94.80000
0.025	0.000000	0.006527	0.549619	0.002566	0.034339	97.00000
0.010	0.000000	0.003330	0.008363	0.000039	0.000474	99.00000
0.005	0.000000	0.046061	0.092117	0.000184	0.004115	99.60000

Table 3.6: Block allocation policies: Machine speeds 3.625 and 3.375

Results

We observe from Tables 3.1 - 3.6 that, for each set of machine speeds, the variation in the summary statistics (excluding the percentage of zero suboptimalities) associated with a change in α is indicative of a direct proportion between the discount rate and the reward suboptimality of policy $u_G(\mathbf{B}^*)$ within the class of block allocation policies. This supports the conclusions of Theorem 3. Generally this trend is most evident in the columns of mean and median suboptimalities. We note that in each of the tables, the percentage of reward suboptimalities that are equal to zero is always positive and tends to 100% as $\alpha \longrightarrow 0$.

3.5 General policies

We now proceed to the more complex problem when the admissible policies $\mathcal{U}$ are general policies. Recall, a general policy u makes allocations at each $\tau \in \bigcup_{m=1}^{M} \{t/s_m, t \in \mathbb{N}\}$. Any bandit which at τ has not been allocated to a machine whose current processing interval goes beyond τ may be allocated at τ to any machine for which τ is a decision epoch. Any such allocations must be made in a non-anticipative way and each of the M machines should always be kept busy at all times. It will simplify matters if we make the following assumption, which will be relaxed later.

Assumption 2

All machine speeds s_m are positive integers.

Recall Corollary 1. A key step in our analysis is the development of lower bounds for the quantities $A(S_j, \alpha)$, $1 \leq j \leq |E| - 1$, presented in Lemma 3. Recall also the numbering of the bandits such that $j(1) > j(2) > \ldots > j(B)$. Use the notational convention $j(0) = |E|$ and consider the range $j(\tilde{b}+1) \leq j \leq j(\tilde{b}) - 1$, $0 \leq \tilde{b} \leq B$. For j in this range the bandits $1, 2, \ldots, \tilde{b}$ can never enter S_j. We now define

$$T^{S_j^c} = \sum_{b=\tilde{b}+1}^{B} T_b^{S_j^c}, \quad j(\tilde{b}+1) \leq j \leq j(\tilde{b}) - 1, \quad 0 \leq \tilde{b} \leq B. \tag{3.27}$$

See (3.15). The proof of Lemma 3 develops that of Lemma 1 (Glazebrook and Wilkinson (2000)) presented in Chapter 2 and describes the novel features required for our more general context.

Lemma 3

When $\mathcal{U}$ is the class of general policies and Assumption 2 holds,

$$A(S_j,\alpha) \;\geq\; e^{-\alpha}\left(\frac{\sum_{m=\tilde{b}+1}^{M} s_m}{\alpha}\right) E\left[\exp\left(\frac{-\alpha T_j^{S_j^c}}{\sum_{m=\tilde{b}+1}^{M} s_m}\right)\right],$$

$$j(\tilde{b}+1) \leq j \leq j(\tilde{b}) - 1, \quad 0 \leq \tilde{b} \leq M-1. \tag{3.28}$$

Proof

Fix general policy u and denote by $\tau_{i,l,m}$, the time of the l^{th} occasion (for any suitable numbering of such occasions) upon which, under control u, machine m processes bandit b in state i. The collection $\left\{T_{i,l}^{S_j^c}, l \in \mathbb{Z}^+\right\}$ is of i.i.d. random variables, all of which share the distribution of $T_i^{S_j^c}$ in Section 3.3.

The aim is to find lower bounds for the quantities

$$\begin{aligned} A(S,\alpha) &= \inf_{u\in\mathcal{U}} \sum_{m=1}^{M} s_m A^{m,u}(S,\alpha) \\ &= \inf_{u\in\mathcal{U}} \sum_{m=1}^{M} s_m \sum_{i\in S} A_i^S\left(\frac{\alpha}{s_m}\right) x_{im}^u(\alpha), \qquad S \subseteq E. \end{aligned} \tag{3.29}$$

From the definitions of the quantities involved

$$\begin{aligned} A(S_j,\alpha) &= \inf_{u\in\mathcal{U}} \sum_{m=1}^{M} s_m \sum_{i\in S_j} A_i^{S_j}\left(\frac{\alpha}{s_m}\right) E_u\left(\sum_{l=1}^{\infty} \int_{\tau_{i,l,m}}^{\tau_{i,l,m}+1/s_m} e^{-\alpha t} dt\right) \\ &= \inf_{u\in\mathcal{U}} \sum_{m=1}^{M} s_m \sum_{i\in S_j} \frac{E\left(\int_0^{T_i^{S_j^c}/s_m} e^{-\alpha t} dt\right)}{\left(\int_0^{1/s_m} e^{-\alpha t} dt\right)} E_u\left(\sum_{l=1}^{\infty} e^{-\alpha\tau_{i,l,m}} \int_0^{1/s_m} e^{-\alpha t} dt\right) \\ &= \inf_{u\in\mathcal{U}} \sum_{m=1}^{M} \sum_{i\in S_j} E_u\left(\sum_{l=1}^{\infty} e^{-\alpha\tau_{i,l,m}}\, s_m \int_0^{T_{i,l}^{S_j^c}/s_m} e^{-\alpha t} dt\right) \\ &\geq \inf_{u\in\mathcal{U}} \sum_{m=1}^{M} \sum_{i\in S_j} E_u\left(\sum_{l=1}^{\infty} e^{-\alpha\tau_{i,l,m}} \int_0^{T_{i,l}^{S_j^c}} e^{-\alpha t} dt\right) \\ &\geq \inf_{u\in\mathcal{U}} \sum_{m=1}^{M} \sum_{i\in S_j} E_u\left(\sum_{l=1}^{\infty} e^{-\alpha}\, e^{-\alpha\lceil\tau_{i,l,m}\rceil} \int_0^{T_{i,l}^{S_j^c}} e^{-\alpha t} dt\right) \\ &= \inf_{u\in\mathcal{U}} \sum_{m=1}^{M} \sum_{i\in S_j} E_u\left(\sum_{l=1}^{\infty} e^{-\alpha} \int_{\lceil\tau_{i,l,m}\rceil}^{T_{i,l}^{S_j^c}+\lceil\tau_{i,l,m}\rceil} e^{-\alpha t} dt\right), \end{aligned} \tag{3.30}$$

where $\lceil x \rceil$ denotes the integer part of x.

Fix $j,\ j(\tilde{b}+1) \leq j \leq j(\tilde{b})-1,\ 0 \leq \tilde{b} \leq M-1$. We now relax the stochastic optimisation problem (3.29). The relaxation takes the form of a single machine scheduling problem with decision epochs $t \in \mathbb{N}$, whose value is a lower bound on $A(S,\alpha)$ as follows; under Assumption 2 we have exactly $\sum_{m=1}^{M} s_m$ decision epochs during time $[t, t+1)$, $t \in \mathbb{N}$. Denote by the pair $(t,r),\ t \in \mathbb{N},\ 0 \leq r \leq \left(\sum_{m=1}^{M} s_m\right) - 1$, the r^{th} decision opportunity afforded during time $[t, t+1)$ in the original problem (3.29). How the r^{th} opportunity is defined is immaterial for reasons which will become clear later. Consider the mapping onto $\mathbb{N}$ given by

$$(t,r) \rightarrow \psi(t,r) \equiv t\left(\sum_{m=1}^{M} s_m\right) + r$$

in which this decision opportunity is thought to occur in the derived single machine problem at time $\psi(t,r)$.

The admissible controls $\grave{\mathcal{U}}$ in the derived single machine problem are non-anticipative and non-idling with certain restrictions placed on the permitted scheduling of bandits $1, 2, \ldots \tilde{b}$. Observe that by construction, these bandits never contribute to the objective in (3.29). In our derived single machine relaxation, bandits $1, 2, \ldots \tilde{b}$ may be scheduled at most $\sum_{m=1}^{\tilde{b}} s_m$ times from the epochs $t(\sum_{m=1}^{M} s_m) \leq r \leq (t+1)(\sum_{m=1}^{M} s_m) - 1$ for each $t \in \mathbb{N}$ and contribute nothing to the corresponding objective. We impose no constraints on the frequency of processing of the remaining bandits. Following the inequality in (3.30), rewards accruing in the single machine problem at decision epochs $\psi(t,r)$ attract discounting as if accruing at time $t \in \mathbb{N}$ in the original problem.

Under this single machine relaxation and from (3.30), it follows that

$$A(S_j,\alpha) \geq \inf_{\grave{u} \in \grave{\mathcal{U}}} \sum_{b=1}^{B} \sum_{t=0}^{\infty} \sum_{r=0}^{(\Sigma_{m=1}^{M} s_m)-1} E_{\grave{u}}\left[\left\{e^{-\alpha}\int_{t}^{t+1} e^{-\alpha s} ds\right\}\left\{\int_{\psi(t,r)}^{\psi(t,r)+1} I_b(s) ds\right\}\right] \tag{3.31}$$

where

$$I_b(s) = \begin{cases} 1, & \text{if bandit } b \text{ is processed at time } s, \text{ having} \\ & \text{paid its first visit to } S_j \text{ at some time } v \leq s, \\ 0, & \text{otherwise.} \end{cases}$$

Inequality (3.31) follows since we have both relaxed the problem and (from (3.30)) reduced the objective. See also, the reasoning following (2.20).

The single machine problem in (3.31) is easily solved. Clearly, since the bandits $1, 2, \ldots, \tilde{b}$ will never contribute to the objective on the righthand side of (3.31) and the function $e^{-\alpha s}$ is decreasing in s, any policy $\grave{u} \in \grave{\mathcal{U}}$ obtaining the infimum in (3.31) should schedule these bandits as often and as early as possible. Hence all decision epochs $\psi(t,r),\ t \in \mathbb{N},\ 0 \leq r \leq (\sum_{m=1}^{\tilde{b}} s_m) - 1$ should be reserved for bandits $1, 2, \ldots, \tilde{b}$ in an optimal policy and we can replace (3.31) by

$$A(S_j, \alpha) \geq \inf_{\grave{u} \in \grave{\mathcal{U}}} \sum_{b=\tilde{b}+1}^{B} \sum_{t=0}^{\infty} \sum_{r=(\Sigma_{m=1}^{\tilde{b}} s_m)}^{(\Sigma_{m=1}^{M} s_m)-1} E_{\grave{u}} \left[\left\{ e^{-\alpha} \int_t^{t+1} e^{-\alpha s} ds \right\} \left\{ \int_{\psi(t,r)}^{\psi(t,r)+1} I_b(s) ds \right\} \right]. \tag{3.32}$$

The right-hand side of (3.32) is a minimisation involving the bandits $b \in \{\tilde{b}+1, \tilde{b}+2, \ldots, B\}$ all of which can enter S_j. It follows from a simple pairwise interchange argument based on the fact that the function $e^{-\alpha s}$ is decreasing in s, that a minimising $\grave{u} \in \grave{\mathcal{U}}$ will make all free choices at the remaining epochs $\{\psi(t,r); t \in \mathbb{N}, \sum_{m=1}^{\tilde{b}} s_m \leq r \leq (\sum_{m=1}^{M} s_m) - 1\}$ to enforce the priority $S_j^c \rightarrow S_j$.

Write

$$T^{S_j^c} = \left\lceil \frac{T^{S_j^c}}{\sum_{m=\tilde{b}+1}^{M} s_m} \right\rceil \sum_{m=\tilde{b}+1}^{M} s_m + R$$

where $0 \leq R \leq (\sum_{m=\tilde{b}+1}^{M} s_m) - 1$. A simple calculation yields

$$\begin{aligned} A(S_j, \alpha) &\geq e^{-\alpha} E \left[\left(\sum_{m=\tilde{b}+1}^{M} s_m - R \right) \int_{\left\lceil \frac{T^{S_j^c}}{\Sigma_{m=\tilde{b}+1}^{M} s_m} \right\rceil}^{\infty} e^{-\alpha s} ds + R \int_{\left\lceil \frac{T^{S_j^c}}{\Sigma_{m=\tilde{b}+1}^{M} s_m} \right\rceil + 1}^{\infty} e^{-\alpha s} ds \right] \\ &= e^{-\alpha} \frac{\left[\left(\sum_{m=\tilde{b}+1}^{M} s_m - R \right) + R e^{-\alpha} \right]}{\alpha} E \left[\exp \left(-\alpha \left\lceil \frac{T^{S_j^c}}{\sum_{m=\tilde{b}+1}^{M} s_m} \right\rceil \right) \right] \\ &\geq e^{-\alpha} \left(\frac{\sum_{m=\tilde{b}+1}^{M} s_m}{\alpha} \right) E \left[\exp \left(\frac{-\alpha T^{S_j^c}}{\sum_{m=\tilde{b}+1}^{M} s_m} \right) \right] \end{aligned}$$

as required. This concludes the proof.

In Theorem 5, $u_G(\mathbf{B}^*)$ is the block allocation policy discussed in Section 3.4.

Theorem 5

Under Assumption 2,

(i) $R^{\text{opt}} = \sum_{m=1}^{M} s_m G_{j(m)}/\alpha + O(1)$; and

(ii) $R^{\text{opt}} - R^{u_G(\mathbf{B}^*)} \leq O(1)$,

where the optimum is taken over the class of general policies.

Proof

From Theorem 1

$$\begin{aligned} R^{\text{opt}} &= \frac{\sum_{m=1}^{M} G_{|E|}^{m}(\alpha)}{\alpha} - \sum_{j=1}^{|E|-1} (G_{j+1} - G_j) \sum_{m=1}^{M} s_m A^{m,\text{opt}}(S_j, \alpha) \\ &\quad + \alpha \sum_{j=1}^{|E|-1} (\delta_{j+1} - \delta_j) \sum_{m=1}^{M} A^{m,\text{opt}}(S_j, \alpha) + O(\alpha), \\ &\leq \frac{\sum_{m=1}^{M} s_m G_{|E|}}{\alpha} + \sum_{m=1}^{M} \delta_{|E|} - \sum_{j=1}^{|E|-1} (G_{j+1} - G_j) A(S_j, \alpha) \\ &\quad + \alpha \sum_{j=1}^{|E|-1} (\delta_{j+1} - \delta_j) \sum_{m=1}^{M} A^{m,\text{opt}}(S_j, \alpha) + O(\alpha). \end{aligned} \tag{3.33}$$

We note that all positive entries in the vector (3.5) are $O(1)$ for all subsets $S \subseteq E$. It follows that

$$\sum_{m=1}^{M} A^{m,\text{opt}}(S, \alpha) = \sum_{m=1}^{M} \sum_{i \in S} A_i^S \left(\frac{\alpha}{s_m} \right) x_{im}^{\text{opt}}(\alpha) = O(1/\alpha)$$

and also that, as expressed in (3.4), the standard Gittins indices are $O(1)$. Hence the quantities G_j and δ_j, $1 \leq j \leq |E|$, are $O(1)$. Using these facts and Lemma 3 within (3.33), we can write

$$\begin{aligned} R^{\text{opt}} &\leq \frac{\sum_{m=1}^{M} s_m G_{|E|}}{\alpha} \\ &\quad - \sum_{\tilde{b}=0}^{M-1} \sum_{j=j(\tilde{b}+1)}^{j(\tilde{b})-1} (G_{j+1} - G_j) \left\{ e^{-\alpha} \left(\frac{\sum_{m=\tilde{b}+1}^{M} s_m}{\alpha} \right) E\left[\exp\left(\frac{-\alpha T^{S_j^c}}{\sum_{m=\tilde{b}+1}^{M} s_m} \right) \right] \right\} + O(1) \\ &= \frac{\sum_{m=1}^{M} s_m G_{|E|}}{\alpha} \\ &\quad - \left(\frac{\sum_{\tilde{b}=0}^{M-1} \sum_{j=j(\tilde{b}+1)}^{j(\tilde{b})-1} (G_{j+1} - G_j) \sum_{m=\tilde{b}+1}^{M} s_m}{\alpha} \right) e^{-\alpha} E\left[\exp\left(\frac{-\alpha T^{S_j^c}}{\sum_{m=\tilde{b}+1}^{M} s_m} \right) \right] + O(1). \end{aligned} \tag{3.34}$$

If we now write

$$e^{-\alpha}E\left[\exp\left(\frac{-\alpha T^{S_j^c}}{\sum_{m=\tilde{b}+1}^{M} s_m}\right)\right] = \left(1-\alpha+O(\alpha^2)\right)\left(1-\alpha E\left[\frac{T^{S_j^c}}{\sum_{m=\tilde{b}+1}^{M} s_m}\right]+O(\alpha^2)\right)$$

$$= 1-\alpha\left(1+E\left[\frac{T^{S_j^c}}{\sum_{m=\tilde{b}+1}^{M} s_m}\right]\right)+O(\alpha^2)$$

in (3.34), we easily conclude that

$$\begin{aligned}
R^{\text{opt}} &\leq \frac{\sum_{m=1}^{M} s_m G_{|E|}}{\alpha} - \left(\frac{\sum_{\tilde{b}=0}^{M-1}\sum_{j=j(\tilde{b}+1)}^{j(\tilde{b})-1}(G_{j+1}-G_j)\sum_{m=\tilde{b}+1}^{M} s_m}{\alpha}\right)+O(1)\\
&= \frac{\sum_{m=1}^{M} s_m G_{|E|}}{\alpha} - \left(\frac{\sum_{\tilde{b}=0}^{M-1}\left\{G_{j(\tilde{b})}-G_{j(\tilde{b}+1)}\right\}\sum_{m=\tilde{b}+1}^{M} s_m}{\alpha}\right)+O(1)\\
&= \frac{\sum_{m=1}^{M} s_m G_{|E|}}{\alpha} - \left(\frac{\sum_{\tilde{b}=0}^{M-1}\sum_{m=\tilde{b}+1}^{M} s_m\left\{G_{j(\tilde{b})}-G_{j(\tilde{b}+1)}\right\}}{\alpha}\right)+O(1)\\
&= \frac{\sum_{m=1}^{M} s_m G_{|E|}}{\alpha} - \left(\frac{\sum_{m=1}^{M} s_m\sum_{\tilde{b}=0}^{m-1}\left\{G_{j(\tilde{b})}-G_{j(\tilde{b}+1)}\right\}}{\alpha}\right)+O(1)\\
&= \frac{\sum_{m=1}^{M} s_m G_{|E|}}{\alpha} - \left(\frac{\sum_{m=1}^{M} s_m\left\{G_{j(0)}-G_{j(m)}\right\}}{\alpha}\right)+O(1)\\
&= \frac{\sum_{m=1}^{M} s_m G_{j(m)}}{\alpha}+O(1). \qquad (3.35)
\end{aligned}$$

But, from the calculations in the proof of Theorem 3 we conclude that

$$R^{u_G(\mathbf{B}^*)} = \frac{\sum_{m=1}^{M} s_m G_{j(m)}}{\alpha}+O(1) \leq R^{\text{opt}}. \qquad (3.36)$$

Theorem 5 follows easily from (3.35) and (3.36).

As in Section 3.4, we finish by concluding from Theorem 5 that $u_G(\mathbf{B}^*)$ enjoys a form of asymptotic optimality in the general class of policies considered here. Not surprisingly, the form of optimality is weaker than in Theorem 4 where attention was restricted to block allocation policies.

Theorem 6 (Asymptotic optimality of block allocation policy $u_G(\mathbf{B}^*)$)

Under Assumption 2, $u_G(\mathbf{B}^*)$ is -1-discount optimal and average-reward optimal in the class of general policies.

3.5.1 Numerical study

The performance of $u_G(\mathbf{B}^*)$ within the class of general policies was investigated more widely by means of a computational study. We used the Value Iteration Algorithm to obtain reward suboptimalities for our block allocation heuristic $u_G(\mathbf{B}^*)$ enabling us to assess the degree of conservatism in our theoretical results.

The Value Iteration Algorithm was used to calculate the expected reward earned when following $u_G(\mathbf{B}^*)$ and also the optimal expected reward from the class of general allocation policies. For the purposes of calculating the optimal expected reward for a particular problem, certain additional features needed to be incorporated into the computer programs. Recall that a general policy makes allocation decisions at each $\tau \in \bigcup_{m=1}^{M} \{t/s_m,\ t \in \mathbb{N}\}$. Any bandit which at τ has not been allocated to a machine whose current processing interval goes beyond τ may be allocated at τ to any machine for which τ is a decision epoch. Hence at each τ, the state of the system must comprise the current state of each bandit, which of the machines have τ as a decision time and finally which, if any, of the bandits are tied to a particular machine at τ and not free for re-allocation then.

Now under Assumption 2, the structure of the set of decision epochs during time $[t, t+1)$ is identical for all $t \in \mathbb{N}$. This evolutionary characteristic can be used to formulate a dynamic program by breaking down each such time block into its component parts, with the arguments at each decision epoch during $[t, t+1)$, $t \in \mathbb{N}$, describing completely the state of the system at that time. Hence each full iteration of the algorithm will consist of a number of steps, though as outlined above, the amount of information required to describe the state of the system completely may be different at each step. Also during time $[t, t+1)$, $t \in \mathbb{N}$, the times of the decision epochs may not be equally spaced, hence the discounting received will vary for each step within each iteration. In order to overcome this, we introduce 'virtual' decision epochs such that each machine has a 'virtual' speed of $s = \text{lcm}(s_1, s_2, \ldots, s_M)$. Hence for each iteration of the algorithm, between $[t, t+1)$, we have s steps representing times $t + (v/s)$, $0 \leq v \leq s-1$. Although there will be no actual transitions at many of these times, the discounting received will be the same at each and at each we must know the state of the system in order to determine the set of possible actions.

We now define the quantities required to fully describe the state of the system at

time $t+(v/s)$, $t \in \mathbb{N}$, $0 \leq v \leq s-1$, under the above formulation. Denote by $\boldsymbol{\beta}$, the B vector whose b^{th} component is the state of bandit b at time $t+(v/s)$ and by $\boldsymbol{\mu}$, the M vector whose m^{th} component is b if bandit b is occupying machine m at time $t+(v/s)$ and 0 if machine m is available for allocation, $1 \leq b \leq B$, $1 \leq m \leq M$. We then define the state of the system at time $t+(v/s)$ as

$$\mathbf{x} = (v, \boldsymbol{\beta}, \boldsymbol{\mu})^{\mathbf{T}}$$

and as in Section 2.6, the set of actions is denoted $A(\mathbf{x}), \mathbf{x} \in \mathbf{X}$, with a a typical member. If action a is chosen at time $t+(v/s)$, when the process is in state $\mathbf{x}$, then the reward earned is

$$r(\mathbf{x}, a) \int_{t+(v/s)}^{t+(v+1)/s} e^{-\alpha u} du = \sum_{m=1}^{M} \left[\frac{r_m(\mathbf{x}, a)}{\int_0^{1/s_m} e^{-\alpha u} du} \right] \int_{t+(v/s)}^{t+(v+1)/s} e^{-\alpha u} du$$

where $r_m(\mathbf{x}, a)$ is the contribution from machine $m, 1 \leq m \leq M$. We write $P_{\mathbf{x}\mathbf{x}'}(a)$ for the probability of a transition from $\mathbf{x}$ to $\mathbf{x}'$ under action a. The optimality equations take the form

$$R^{\text{opt}}_{t+(v+1)/s}(\mathbf{x}) = \max_{a \in A(\mathbf{x})} \left\{ r(\mathbf{x}, a) \int_0^{1/s} e^{-\alpha u} du + e^{-\alpha/s} \sum_{\mathbf{x}' \in \mathbf{X}} P_{\mathbf{x}\mathbf{x}'}(a) R^{\text{opt}}_{t+(v/s)}(\mathbf{x}') \right\}$$

with

$$R^{\text{opt}}_0(\mathbf{x}) = \max_{a \in A(\mathbf{x})} \left\{ r(\mathbf{x}, a) \int_0^{1/s} e^{-\alpha u} du \right\} \qquad \mathbf{x} \in \mathbf{X}.$$

We exit the dynamic program when $R^{\text{opt}}_{t+(v+1)/s}(\mathbf{x}) - R^{\text{opt}}_{t+(v/s)}(\mathbf{x}) \leq 10^{-7}$, $t = 0, 1, 2, \ldots$, $0 \leq v \leq s-1$.

Broadly speaking the computer programs given in Appendices B, C and D are formulated in this way. As in Section 3.4.1, the system studied is a four-armed bandit problem, each arm with four states. Two machines with speeds s_1 and s_2, are available for processing. For given values of the discount rate α and the machine speeds s_1 and s_2, the computer programs generate four-armed bandit problems with random reward vectors generated from a uniform $[l, u]$, for given l and u, a random starting state $\mathbf{k}$ and random irreducible transition matrices. Initially the entries in each transition matrix are generated from a uniform (0.1,0.9) then the matrices are normalised such that rows sum to 1. Since all entries of the resultant matrices are positive, the irreducibility assumed in Section 3.3 follows.

For each α from $\{0.5, 0.25, 0.1, 0.05, 0.025, 0.01, 0.005\}$ and (recalling Assumption 2) sets of speeds (s_1, s_2) from $\{(6,1), (5,2), (4,3)\}$, 500 four-armed bandit problems were investigated where $r_i^b \sim U(2,5)$. The results of the study are given in Tables 3.7 - 3.9. Each entry gives summary statistics (as outlined in Section 3.4.1) for the 500 simulated suboptimalities (values of $R^{\text{opt}} - R^{u_G(\mathbf{B}^*)}$).

Alpha	Minimum	Median	Maximum	Mean	Std Dev	Percent
0.500	0.000000	0.094460	2.762422	0.131243	0.232093	31.40000
0.250	0.000000	0.085834	1.086985	0.082941	0.159201	47.60000
0.100	0.000000	0.062988	0.636120	0.035168	0.087571	65.40000
0.050	0.000000	0.031613	0.518866	0.012094	0.044904	78.20000
0.025	0.000000	0.010084	0.614066	0.003737	0.032695	92.20000
0.010	0.000000	0.004355	0.808606	0.002523	0.037582	95.80000
0.005	0.000000	0.004252	0.040504	0.000162	0.002325	99.00000

Table 3.7: General Allocation Policies: Machine speeds 6 and 1

Alpha	Minimum	Median	Maximum	Mean	Std Dev	Percent
0.500	0.000000	0.085616	1.073233	0.085356	0.162720	45.00000
0.250	0.000000	0.043794	0.979615	0.037045	0.102134	59.40000
0.100	0.000000	0.027737	0.695575	0.013388	0.056590	78.60000
0.050	0.000000	0.007823	0.899125	0.006229	0.053615	90.20000
0.025	0.000000	0.006262	0.047858	0.000769	0.004318	92.60000
0.010	0.000000	0.001757	0.148622	0.000386	0.006697	97.40000
0.005	0.000000	0.001100	0.004034	0.000015	0.000205	99.00000

Table 3.8: General Allocation Policies: Machine speeds speeds 5 and 2

Results

We observe from Tables 3.7 - 3.9 that the variation in the summary statistics (excluding percentage of zero suboptimalities) associated with a change of discount rate is consistent with an $O(\alpha)$ bound on the degree of reward suboptimality of $u_G(\mathbf{B}^*)$

Alpha	Minimum	Median	Maximum	Mean	Std Dev	Percent
0.500	0.000000	0.062232	0.840281	0.058620	0.121182	52.80000
0.250	0.000000	0.029919	0.772910	0.022742	0.081283	71.60000
0.100	0.000000	0.020747	0.528554	0.008159	0.039364	83.60000
0.050	0.000000	0.009014	0.685113	0.003789	0.035363	91.20000
0.025	0.000000	0.005724	0.290583	0.001299	0.016445	96.40000
0.010	0.000000	0.000798	0.231192	0.000473	0.010330	98.80000
0.005	0.000000	0.001407	0.001407	0.000003	0.000063	99.80000

Table 3.9: General Allocation Policies: Machine speeds 4 and 3

within the class of general policies. These results suggest a degree of conservatism in the suboptimality bound in Theorem 5 (ii) obtained from the primal dual theory.

The reason for this suggested conservatism may be that by focussing on specific models in the numerical investigation, the very general claims made about wide classes of systems in the theory appear more conservative. The computational study may inadvertently be confined to a sub-class of problems for which the bound on the degree of reward suboptimality is tighter than for the more general class of allocation policies. A more straightforward explanation is that the bounds on the rewards derived from the theory, leading to an $O(1)$ suboptimality bound, may not be sufficiently tight.

Note that, as would be expected, the computational results from Sections 3.4.1 and 3.5.1 indicate that the bound on the degree of reward suboptimality of policy $u_G(\mathbf{B}^*)$ is tighter when restricting to the class of block allocation policies than when considering the much less restrictive general class of policies. This is seen most clearly by comparing Tables 3.1, 3.2 and 3.4 with Tables 3.7, 3.8 and 3.9 respectively. This effect is surprisingly mild, however.

3.5.2 On relaxing Assumption 2

We now relax the requirement of Assumption 2 that $s_m \in \mathbb{Z}^+$, $1 \leq m \leq M$. Suppose now that the s_m's are all positive rational numbers and for some $K \in \mathbb{Z}^+$ we have $Ks_m \in \mathbb{Z}^+$, $1 \leq m \leq M$. We consider the problem in which the machine speeds are all increased by a factor K. By applying the above theory to this problem we infer from

Theorem 5 that

$$R_K^{\text{opt}} = \left\{ \frac{K \sum_{m=1}^{M} s_m G_{j(m)}}{\alpha} \right\} + O(1), \tag{3.37}$$

and

$$R_K^{\text{opt}} - R_K^{u_G(\mathbf{B}^*)} \leq O(1), \tag{3.38}$$

where subscript K denotes the speed up factor used. Since the effect of the speed up is to increase returns under the average reward criterion by a factor K we can easily deduce from (3.37) and (3.38) that

$$R^{\text{opt}} = \left\{ \frac{\sum_{m=1}^{M} s_m G_{j(m)}}{\alpha} \right\} + O(1),$$

and

$$R^{\text{opt}} - R^{u_G(\mathbf{B}^*)} \leq O(1).$$

Hence Theorems 5 and 6 continue to hold when $s_m \in \mathbb{Q}^+$, $1 \leq m \leq M$. To deduce these results for general $s_m \in \mathbb{R}^+$, $1 \leq m \leq M$, is not straightforward.

3.6 Gittins index policies

In this section, we discuss the performance of the Gittins index policy when implemented in the parallel machine environment with machines operating at different speeds. Denote this policy u_G. Define $m(\tau)$, $\tau \in \bigcup_{m=1}^{M} \{t/s_m,\ t \in \mathbb{N}\}$ as the number of machines for which τ is a decision epoch. Under policy u_G, at each τ, the $m(\tau)$ bandits with the maximal current Gittins indices, among those bandits not tied to a machine at τ, are chosen for processing. The allocation of bandits to machines is made to match the bandits with the maximal current indices to the available machines operating at the greatest speeds.

We begin our discussion of u_G by examining a class of multi-armed bandit problems modelled as in Section 3.2 but with certain restrictions placed on the structure and number of bandits. In the systems we consider the number of bandits available for processing satisifies $B = 2M$ and the following conditions hold;

$$\begin{aligned} \min_{M+1 \leq b \leq B} G_{k(b)} > \max_{j \in E_1} G_j \geq \min_{j \in E_1} G_j > \max_{j \in E_2} G_j \geq \min_{j \in E_2} G_j > \dots \\ \dots \geq \min_{j \in E_M} G_j > \max_{j \in \bigcup_{b=M+1}^{B} \{E_b \setminus k(b)\}} G_j \end{aligned} \tag{3.39}$$

and

$$P^b_{k(b)k(b)} = 0, \quad M+1 \leq b \leq B, \tag{3.40}$$

where in (3.39) and (3.40) bandits are numbered, as in Section 3.4, such that $j(1) > j(2) > \ldots > j(B)$ and the initial state of bandit b is $k(b)$.

We now re-number the bandits $M+1, M+2, \ldots, B$ as $1', 2', \ldots, M'$ such that

$$G_{k(1')} \geq G_{k(2')} \geq \ldots \geq G_{k(M')}. \tag{3.41}$$

It follows from (3.39), (3.40) and (3.41) that at time 0 policy u_G will allocate bandit m' to machine m, $1 \leq m \leq M$. Then, following a single transition in machine m, bandit m' will certainly be in a state within $S_{j(M)}$ and therefore will never be processed again under u_G. The subsequent scheduling according to u_G will allocate bandit m to machine m from time $1/s_m$ onwards, $1 \leq m \leq M$. Hence u_G is equivalent to a block allocation policy as described in Section 3.4. The equivalent block allocation policy, $u_G(\hat{\mathbf{B}})$ say, is determined by the partition

$$\hat{B}_m = \{m, m'\}, \quad 1 \leq m \leq M.$$

Utilising Corollary 2 and following the reasoning in the proof of Theorem 2 leading to (3.20), we obtain

$$\begin{aligned}
R^{u_G(\hat{\mathbf{B}})} &= \frac{\sum_{m=1}^M s_m G_{j(m)}}{\alpha} + \sum_{m=1}^M \sum_{j=j(\hat{B}_m)}^{|E|-1} (G_{j+1} - G_j)\, E\left(T^{S_j^c}_{\hat{B}_m}\right) + \sum_{m=1}^M \delta_{j(\hat{B}_m)} + O(\alpha) \\
&= \frac{\sum_{m=1}^M s_m G_{j(m)}}{\alpha} + \sum_{m=1}^M \sum_{j=j(m)}^{|E|-1} (G_{j+1} - G_j)\, E\left(T^{S_j^c}_{m}\right) \\
&\qquad + \sum_{m=1}^M \delta_{j(m)} + \sum_{m=1}^M \sum_{j=j(m)}^{|E|-1} (G_{j+1} - G_j)\, E\left(T^{S_j^c}_{m'}\right) + O(\alpha) \\
&= R^{u_G(\mathbf{B}^*)} - \sum_{m=1}^M \sum_{j=j(M)}^{j(m)-1} (G_{j+1} - G_j)\, E\left(T^{S_j^c}_{m'}\right) + O(\alpha).
\end{aligned} \tag{3.42}$$

By (3.39), the second term on the right-hand side of (3.42) is strictly positive. It follows that

$$R^{u_G(\mathbf{B}^*)} - R^{u_G(\hat{\mathbf{B}})} \geq O(1)$$

and hence

$$R^{u_G(\mathbf{B}^*)} - R^{u_G} \geq O(1).$$

Utilising Theorem 5 (ii) we infer that, when $B = 2M$ and conditions (3.39) and (3.40) hold,

$$R^{\text{opt}} - R^{u_G} \geq O(1), \tag{3.43}$$

where the optimum is taken over the class of general policies.

Remark

From the above discussion it follows that when we restrict to the class of block allocation policies, an $O(1)$ bound on the degree of reward suboptimality of policy u_G is the strongest assertion that can be made. Recall from Theorem 3 that our block allocation policy $u_G(\mathbf{B}^*)$ comes within an $O(\alpha)$ quantity of optimality in the classs of block allocation policies.

An immediate consequence of (3.43) is that for a general system, modelled as in Section 3.2, the strongest assertion that could possibly be made concerning the performance of the Gittins index policy u_G when compared with an optimal general policy is

$$R^{\text{opt}} - R^{u_G} \leq O(1). \tag{3.44}$$

3.6.1 Numerical study

Further examination of the performance of u_G was conducted via computational studies. The computer programs used to perform the studies are given in Appendices E, F and G. The systems simulated are four armed bandit problems, each arm with four states, on two machines operating at varying speeds (hence $B = 2M$). For each α from the set $\{0.5,0.25,0.1,0.05,0.025,0.01,0.005\}$ and set of speeds (s_1, s_2) from $\{(6,1),(5,2),(4,3)\}$, 1000 four-armed bandit problems were investigated where

$$r^{1'}_{k(1')} \sim U(7.5, 8.0) \qquad r^{1'}_{i} \sim U(0.00001, 0.00002), \quad i \in \{E_{1'} \setminus k(1')\}$$

$$r^{2'}_{k(2')} \sim U(6.5, 7.0) \qquad r^{2'}_{i} \sim U(0.00001, 0.00002), \quad i \in \{E_{2'} \setminus k(2')\}$$

$$r^{1}_{i} \sim U(5.0, 5.5), \quad i \in \{E_1\}$$

$$r^{2}_{i} \sim U(4.0, 4.5), \quad i \in \{E_2\}$$

$$P^{1'}_{k(1')k(1')} = 0 \qquad P^{1'}_{ik(1')} \sim U(0.00001, 0.00002), \quad i \in \{E_{1'} \setminus k(1')\}$$

$$P^{2'}_{k(2')k(2')} = 0 \qquad P^{2'}_{ik(2')} \sim U(0.00001, 0.00002), \quad i \in \{E_{2'} \setminus k(2')\}$$

and the irreducible transition matrices for bandits 1 and 2 were generated as in Section 3.5.1. The above set-up ensures that conditions (3.39) and (3.40) are satisfied, as well as the model assumptions of Section 3.3. The Value Iteration Algorithm was used to calculate the expected reward earned when following the Gittins index policy u_G outlined earlier and also the optimal expected reward for a general allocation policy. The tolerance ϵ as in (2.37) was 10^{-7}. The results of the study are given in Tables 3.10 - 3.12. Each entry gives summary statistics for the simulated suboptimalities $R^{\text{opt}} - R^{u_G}$, where the optimum is taken over the class of general policies.

Note that in the computer programs given in appendices E, F and G, we also calculated the reward from policy $u_G(\mathbf{B}^*)$ but in almost every case, this latter reward was optimal. These results are not included in the tables.

Alpha	Minimum	Median	Maximum	Mean	Std Dev	Percent
0.500	0.000000	0.083563	0.311930	0.053301	0.069297	44.00000
0.250	0.160707	0.445158	0.783677	0.447951	0.104763	0.000000
0.100	0.459886	0.761557	1.062492	0.761561	0.099588	0.000000
0.050	0.521163	0.882986	1.197457	0.880535	0.103183	0.000000
0.025	0.617559	0.941002	1.219683	0.939476	0.100997	0.000000
0.010	0.662790	0.972263	1.279945	0.974466	0.104614	0.000000
0.005	0.629853	0.987003	1.272824	0.987534	0.104151	0.000000

Table 3.10: Gittins index policies: Machine speeds 6 and 1

Alpha	Minimum	Median	Maximum	Mean	Std Dev	Percent
0.500	0.153979	0.443928	0.752815	0.447557	0.101421	0.000000
0.250	0.451515	0.706026	0.979331	0.711194	0.102019	0.000000
0.100	0.531622	0.879605	1.168430	0.878169	0.107024	0.000000
0.050	0.597172	0.937872	1.209002	0.939852	0.101145	0.000000
0.025	0.685041	0.968469	1.272876	0.969988	0.104461	0.000000
0.010	0.688725	0.991660	1.246305	0.991629	0.105844	0.000000
0.005	0.710585	0.993276	1.297626	0.994826	0.105020	0.000000

Table 3.11: Gittins index policies: Machine speeds 5 and 2

Alpha	Minimum	Median	Maximum	Mean	Std Dev	Percent
0.500	0.469605	0.632621	0.930688	0.640273	0.090568	0.000000
0.250	0.598597	0.791821	0.995329	0.799341	0.092868	0.000000
0.100	0.693229	0.893028	1.100247	0.904046	0.090140	0.000000
0.050	0.826656	0.977319	1.276992	0.984876	0.092932	0.000000
0.025	0.848198	0.996657	1.297762	1.005589	0.093392	0.000000
0.010	0.708372	0.990995	1.295671	0.992747	0.104965	0.000000
0.005	0.663211	0.997208	1.255686	0.990440	0.104698	0.000000

Table 3.12: Gittins index policies: Machine speeds 4 and 3

Results

The results in Tables 3.10 - 3.12 give a clear indication that, under the stated conditions, the bound on the degree of reward suboptimality of the Gittins index policy is $O(1)$ and hence the assertion in (3.43) is borne out.

3.7 On relaxing Assumption 1

We now consider the relaxation of Assumption 1. Hence we must now contemplate the possibility that two distinct members $j,\ k$ of E have $G_j = G_k,\ g_j = g_k$ and hence from (3.4) have associated Gittins indices within an $O(\alpha^2)$ quantity of each other. We proceed as follows: Let

$$E = \bigcup_{n=1}^{N} E_n$$

be a partition of E such that

$$j,\ k \in E_n \Rightarrow G_j = G_k,\ g_j = g_k,\ 1 \leq n \leq N \tag{3.45}$$

and suppose that the E_n's are maximal in this respect. We use $G(E_n)$, $g(E_n)$ to denote respectively the common values of $G_j,\ g_j$ for $j \in E_n,\ 1 \leq n \leq N$ and re-number the E_n such that

$$n_1 > n_2 \Rightarrow \text{ either } G(E_{n_1}) > G(E_{n_2}) \text{ or } G(E_{n_1}) = G(E_{n_2}) \text{ and } g(E_{n_1}) > g(E_{n_2}).$$

From (3.4) and (3.45) it follows that there must exist α^* such that for all $\alpha \in (0, \alpha^*]$, the ordering $E_N \rightarrow E_{N-1} \rightarrow \ldots \rightarrow E_2 \rightarrow E_1$ on the partition sets is consistent with a

Gittins index ordering on all machines m, $1 \leq m \leq M$. We shall suppose henceforth that $\alpha \in (0, \alpha^*]$. We write $S_n = \bigcup_{v=1}^{n} E_v$ for the subset of E of cardinality $\sum_{v=1}^{n} |E_v|$ with lowest identifiers and extend the ordering of partition sets to an ordering of elements of E as follows: denote

$$\left[n, j_m^{\alpha}\right], \qquad j \in \{1, 2, \ldots, |E_n|\},\ 1 \leq n \leq N,\ 1 \leq m \leq M,$$

as the member of E_n, $1 \leq n \leq N$, with the j^{th} largest index. Now number the members of E such that

$$\left[N, |E_N|_m^{\alpha}\right] \longrightarrow \left[N, (|E_N| - 1)_m^{\alpha}\right] \longrightarrow \ldots \longrightarrow \left[1, 2_m^{\alpha}\right] \longrightarrow \left[1, 1_m^{\alpha}\right].$$

Now let $u_{\tilde{G}}(\mathbf{B})$ be a policy constructed as in Section 3.3, but where Assumption 1 is relaxed. We write $n(b)$ for the number of the partition set with the minimal identifier containing any state belonging to b, then number the bandits such that

$$n(1) \geq n(2) \geq \ldots \geq n(B).$$

We also write

$$n(B_m) = \max\{n(b)\,;\, b \in B_m\}, \qquad 1 \leq m \leq M,$$

for the maximal identifier in the collection of bandits B_m. Recall the definition of δ prior to Theorem 1. Using a simple elaboration of the analysis leading to Corollary 2, we have, in an obvious notation, that

$$R^{u_{\tilde{G}}(\mathbf{B})} = \sum_{m=1}^{M} \frac{s_m G(E_{n(B_m)})}{\alpha} + \sum_{m=1}^{M} \sum_{n=n(B_m)}^{N} \{G(E_{n+1}) - G(E_n)\}\, E\big(T_{B_m}^{S_n^c}\big) + \sum_{m=1}^{M} \delta(E_{n(B_m)}) + O(\alpha).$$

From a development of the proof of Theorem 3, we infer that any partition maximising $R^{u_{\tilde{G}}(\mathbf{B})}$ over choices of partition $\mathbf{B}$, will be of the form

$$B_m^* = \begin{cases} \{m\}, & 1 \leq m \leq M-1, \\ \{M, M+1, \ldots, B\}, & m = M \end{cases}$$

and is guaranteed to bring us within an $O(\alpha)$ quantity of optimality in the class of block allocation policies.

A development of the proof of Theorem 5 is also available for the analysis of $u_{\tilde{G}}(\mathbf{B})$. It follows that $R^{u_{\tilde{G}}(\mathbf{B}^*)}$ comes within an $O(1)$ quantity of the optimum within the class of general policies. Versions of Theorem 4 and 6 then follow. We summarise these conclusions in the following result:

Theorem 7

(i) When $\mathcal{U}$ is the class of block allocation policies

$$R^{\text{opt}} - R^{u_{\tilde{G}}(\mathbf{B}^*)} \leq O(\alpha).$$

Any such policy $u_{\tilde{G}}(\mathbf{B}^*)$ is 0-discount optimal, average-overtaking optimal and average-reward optimal within the class.

(ii) When $\mathcal{U}$ is the general class of policies

$$R^{\text{opt}} - R^{u_{\tilde{G}}(\mathbf{B}^*)} \leq O(1).$$

Any such policy $u_{\tilde{G}}(\mathbf{B}^*)$ is -1-discount optimal and average-reward optimal within the class.

Plainly, without Assumption 1 we cannot assert Blackwell optimality for $u_{\tilde{G}}(\mathbf{B}^*)$ under the conditions of Theorem 2 since it is not now guaranteed that the scheduling on each machine is a Gittins index ordering.

Chapter 4

Bandit Problems on Parallel Machines with stochastic machine availability

4.1 Introduction

This chapter considers generalisations of the discounted branching bandit and the discounted multi-armed bandit problems on parallel machines, to the case where the collection of machines available for processing is itself a stochastic process. Stochastic scheduling models incorporating machine breakdowns have been studied by many authors since Glazebrook (1984), (1987) demonstrated that index policies remain optimal for a range of extensions of the classical multi-armed bandit model in which the single machine providing service is subject to breakdown and repair. See for example Birge *et al.* (1990) and Pinedo and Rammouz (1988). The aim of this chapter is to develop the tools of analysis rooted in the achievable region approach, as described in Chapter 2, in such a way as to facilitate analysis of heuristics for scheduling control in contexts where the set of machines available for processing evolves as a (reasonably general) stochastic process.

Section 4.2 concerns our modelling of machine availability. Roughly speaking, we shall simply require of the machine availability process that the long-run proportion of epochs at which all M machines are available is positive. Examples demonstrate that the technical conditions concerned are satisfied by a range of standard models.

The scheduling models which concern us are described Section 4.3. One (Model 1) is a discounted branching bandit while the other (Model 2) is a discounted multi-armed bandit. In both cases, service is provided by M machines which are identical in their processing capacity, but which are not always available for processing. In Section 4.4 we develop the tools of analysis which are designed to give information on the degree of suboptimality of heuristic scheduling controls. These tools are utilised in Section 4.5 (Model 1) and Section 4.6 (Model 2) to evaluate the performance of Gittins index-based scheduling controls for our problems with intermittent machine availability. These simple index-based heuristics partition the jobs to be scheduled among the (potentially available) machines at time 0. Thereafter, the individual machines process according to index policies whenever they are up. In some cases the rewards from these simple controls come within $O(1)$ of optimality, while in others we can achieve $O(\alpha)$. Much practical and theoretical interest attaches to problems with small discount rate α and to the related limit $\alpha \to 0$. In Sections 4.5 and 4.6, various forms of asymptotic optimality are claimed for limit policies derived from our Gittins index based controls and numerical investigations allow insights concerning the degree of conservatism in the theoretical results.

4.2 Machine Availability

We have a maximum of $M < \infty$ identical machines which we label $\{1, 2, \ldots, M\}$. At each $t \in \mathbb{N}$, a collection of machines, M_t, are available for the scheduling of stochastic jobs or bandits. We shall suppose that the stochastic process $\{M_t, t \in \mathbb{N}\}$, taking values in $2^{\{1,2,\ldots,M\}}$, is independent of the scheduling control and of the system state process. In order to proceed we shall need a range of numerical descriptors of the machine availability process $\{M_t, t \in \mathbb{N}\}$.

We begin our introduction of processes related to machine availability by introducing $2M$ derived indicator processes. We firstly define $\{I_{m,t}, t \in \mathbb{N}\}$, $1 \leq m \leq M$, by

$$I_{m,t} = \begin{cases} 1, & \text{when } m \in M_t, \\ 0, & \text{otherwise} \end{cases}$$

and in a similar manner we define $\{J_{m,t},\, t \in \mathbb{N}\}, 1 \leq m \leq M$, by

$$J_{m,t} = \begin{cases} 1, & \text{when } |M_t| \geq m, \\ 0, & \text{otherwise.} \end{cases}$$

Hence $I_{m,\cdot}$ marks when machine m is available for processing, while $J_{m,\cdot}$ indicates when the number of machines at our disposal is at least m. It is clear that

$$|M_t| = \sum_{m=1}^{M} I_{m,t} = \sum_{m=1}^{M} J_{m,t}, \qquad t \in \mathbb{N}.$$

We now develop continuous time extensions of these discrete time processes. Define $\{M(s), s \in \mathbb{R}^+\}$, $\{I_m(s), s \in \mathbb{R}^+\}$, $1 \leq m \leq M$, and $\{J_m(s), s \in \mathbb{R}^+\}$, $1 \leq m \leq M$, by

$$\begin{aligned} M(s) &= |M_t|, \qquad s \in [t, t+1),\ t \in \mathbb{N}, \\ I_m(s) &= I_{m,t}, \qquad s \in [t, t+1),\ t \in \mathbb{N}, \\ J_m(s) &= J_{m,t}, \qquad s \in [t, t+1),\ t \in \mathbb{N}. \end{aligned}$$

We continue our account of the key quantities relating to machine availability by introducing the *cumulative availability processes* $\{\bar{M}(s), s \in \mathbb{R}^+\}$, $\{\bar{I}_m(s), s \in \mathbb{R}^+\}$, $1 \leq m \leq M$, and $\{\bar{J}_m(s), s \in \mathbb{R}^+\}$, $1 \leq m \leq M$, defined by

$$\begin{aligned} \bar{M}(s) &= \int_0^s M(u)\, du, \qquad s \in \mathbb{R}^+, \\ \bar{I}_m(s) &= \int_0^s I_m(u)\, du, \qquad s \in \mathbb{R}^+, \\ \bar{J}_m(s) &= \int_0^s J_m(u)\, du, \qquad s \in \mathbb{R}^+. \end{aligned}$$

The cumulative processes defined above are all non-decreasing, piecewise linear and continuous on all sample paths. It is clear that

$$\bar{M}(s) = \sum_{m=1}^{M} \bar{I}_m(s) = \sum_{m=1}^{M} \bar{J}_m(s), \qquad s \in \mathbb{R}^+$$

and all are expressions for the accumulated machine availability up to time s. We now define the inverse process $\{\bar{M}^{-1}(s),\, s \in \mathbb{R}^+\}$, by

$$\bar{M}^{-1}(s) = \inf_{t \in \mathbb{R}^+} \{t;\, \bar{M}(t) > s\}, \qquad s \in \mathbb{R}^+, \tag{4.1}$$

which has the interpretation as the time required to achieve an accumulated machine availability equal to s. The process $\bar{M}^{-1}(s)$ will also be non-decreasing and piecewise

linear, but will have jump discontinuities corresponding to epochs at which $|M_t| = 0$. We also define the inverse processes $\{\bar{I}_m^{-1}(s),\ s \in \mathbb{R}^+\},\ 1 \leq m \leq M,$ and $\{\bar{J}_m^{-1}(s),\ s \in \mathbb{R}^+\},\ 1 \leq m \leq M$, given by

$$\begin{aligned} \bar{I}_m^{-1}(s) &= \inf_{t \in \mathbb{R}^+} \{t\,;\, \bar{I}_m(t) > s\}, \qquad 1 \leq m \leq M,\ s \in \mathbb{R}^+, \\ \bar{J}_m^{-1}(s) &= \inf_{t \in \mathbb{R}^+} \{t\,;\, \bar{J}_m(t) > s\}, \qquad 1 \leq m \leq M,\ s \in \mathbb{R}^+. \end{aligned}$$

We conclude our discussion of measures of machine availability by refining previously defined processes. Let μ be an integer in the range $\{0, 1, 2, \ldots, M\}$. We develop the process $\{\bar{M}_\mu(s),\ s \in \mathbb{R}^+\}$ by

$$\bar{M}_\mu(s) = \int_0^s \{M(u) - \mu\}^+ du,$$

where

$$x^+ = \begin{cases} x, & \text{if } x \geq 0, \\ 0, & \text{otherwise.} \end{cases}$$

The inverse process corresponding to $\{\bar{M}_\mu^{-1}(s),\ s \in \mathbb{R}^+\}$ is defined as in (4.1). These measures are needed in situations where μ machines are 'reserved'.

We now fix time $t \in \mathbb{R}^+$ and develop cumulative processes $\{\bar{I}_m(s;t),\ s \in \mathbb{R}^+\},\ 1 \leq m \leq M$ and $\{\bar{J}_m(s;t),\ s \in \mathbb{R}^+\},\ 1 \leq m \leq M,$ starting from t. Hence we have

$$\begin{aligned} \bar{I}_m(s;t) &= \bar{I}_m(s+t) - \bar{I}_m(t), \qquad s \in \mathbb{R}^+, \\ \bar{J}_m(s;t) &= \bar{J}_m(s+t) - \bar{J}_m(t), \qquad s \in \mathbb{R}^+. \end{aligned}$$

For each $t \in \mathbb{R}^+$ the corresponding inverse process $\{\bar{I}_m^{-1}(s;t),\ s \in \mathbb{R}^+\},\ 1 \leq m \leq M,$ is given by

$$\bar{I}_m^{-1}(s;t) = \inf_{u \in \mathbb{R}^+} \{u\,;\, \bar{I}_m(u;t) > s\}, \qquad s \in \mathbb{R}^+,$$

with $\bar{I}_m^{-1}(s;t)$ having the interpretation as the time required from t until machine m's total availability exceeds s. We define $\{\bar{J}_m^{-1}(s;t),\ s \in \mathbb{R}^+\},\ 1 \leq m \leq M,$ similarly.

We now state an important technical condition which is required at key points in the development of the subsequent theory. In its statement we use $\{\mathcal{H}_t,\ t \in \mathbb{N}\}$ for the history of the machine availability process $\{M_t,\ t \in \mathbb{N}\}$ up to time t. Think of $\mathcal{H}_t$ as the sequence of all the states of the machine process up to time t. Recall the process $\bar{M}^{-1}(s)$, which has the interpretation as the time required to achieve an accumulated machine availability equal to s. It follows that

$$\bar{M}^{-1}(s) \geq \frac{s}{M} \to \infty, \qquad s \to \infty$$

and we shall be concerned to ensure that the divergence of $\bar{M}^{-1}(s)$ is not worse than linear in s. The appropriate condition is given as Condition 1 below with Condition 1′ as a simpler alternative.

Condition 1

There exist constants a and b such that

$$E\{\bar{J}_M^{-1}(s;t) \,|\, \mathcal{H}_t\} \leq a + bs, \qquad s \in \mathbb{R}^+, \text{ w.p. 1}, \tag{4.2}$$

for all choices of $t \in \mathbb{N}$.

If (4.2) is only required to hold for the case $t = 0$, we shall refer to Condition 1′.

Condition 1′

There exist constants a and b such that

$$E\{\bar{J}_M^{-1}(s) \,|\, \mathcal{H}_0\} \leq a + bs, \qquad s \in \mathbb{R}^+, \text{ w.p. 1}. \tag{4.3}$$

Note that, from the definitions of the quantities concerned, it is always true that $M(s) \geq MJ_M(s)$, $s \in \mathbb{R}^+$. Inequality (4.3) is then seen to imply that

$$E\{\bar{M}^{-1}(s) \,|\, \mathcal{H}_0\} \leq a + \left(\frac{b}{M}\right) s, \qquad s \in \mathbb{R}^+, \text{ w.p. 1}.$$

We now give examples of machine availability processes for which Condition 1 or 1′, as appropriate, hold.

Example 1 (Irreducible Markov Chain)

We suppose that the process of the number of machines available $\{|M_t|,\, t \in \mathbb{N}\}$ is an irreducible (and hence positive recurrent) Markov chain. If $|M_t| = m$, we write T_{mM} as the time from t until the first occasion upon which all M machines are available and $T_{MM}(\nu)$, $1 \leq \nu \leq \infty$, for successive first return times to this state. The parameters μ_{mM} and μ_{MM} are respectively the corresponding (finite) expectations of these random times. We observe that when $s \in \mathbb{N}$,

$$\begin{aligned} E\{\bar{J}_M^{-1}(s;t) \,|\, |M_t| = m\} &= E\left(T_{mM} + \sum_{\nu=1}^{s} T_{MM}(\nu)\right) \\ &= \mu_{mM} + s\mu_{MM} \\ &\leq \left(\max_{1 \leq m \leq M} \mu_{mM}\right) + s\mu_{MM}. \end{aligned} \tag{4.4}$$

From (4.4) and the Markovian structure, it is clear that Condition 1 is satisfied.

Example 2 (Breakdown/repair process)

We suppose that each of the M potentially available machines has a breakdown/repair process which is independent of all other machines and that the breakdown and repair rates for machine m are equal to β_m and ρ_m respectively in the sense that

$$\begin{aligned} p(I_{m,t+1} = 0 \,|\, I_{m,t} = 1) &= \beta_m, \qquad t \in \mathbb{N}, \\ p(I_{m,t+1} = 1 \,|\, I_{m,t} = 0) &= \rho_m, \qquad t \in \mathbb{N}, \end{aligned}$$

for $1 \leq m \leq M$. If $0 < \beta_m, \rho_m < 1$, $1 \leq m \leq M$ then $\{M_t, t \in \mathbb{N}\}$ is an irreducible Markov chain. That Condition 1 is satisfied follows from an analysis similar to that applied in Example 1.

Example 3 ($|M_t|$ non-decreasing)

Consider any process $\{|M_t|, t \in \mathbb{N}\}$ for which $|M_t|$ is non-decreasing almost surely. We define the first occasion at which all M machines are available for processing as

$$X_M = \inf_{t \in \mathbb{R}} \{t; |M_t| = M\}.$$

Plainly, in this case,

$$\bar{J}_M^{-1}(s) = s + X_M.$$

Condition 1′ will be satisfied if $E(X_M \,|\, \mathcal{H}_0)$ is uniformly bounded. Please note that, henceforth, we shall *not* make conditioning upon $\mathcal{H}_0$ explicit in the notation.

4.3 The Models

We shall consider two models for the scheduling of stochastic jobs or bandits on a collection of M machines. Machine availability is as described at the beginning of Section 4.2.

4.3.1 Model 1: discounted branching bandit

We consider an extension of the branching bandit problem, as in Section 1.4, to the parallel machine case where the collection of machines available is a stochastic process as described in Section 4.2. As in Section 1.4, jobs are classified in one of $C < \infty$ classes, labelled $\{1, 2, \ldots, C\}$, though we now assume a unit service time for each job class.

Jobs are chosen for processing at each $t \in \mathbb{N}$, with (at most) one being allocated to each available machine. We define the state of the system $\mathbf{N}(t)$ as in Section 1.4 but for the discrete time case. If a class i job is chosen for processing at time t on machine m, then at time $t+1$ that class i job disappears to be replaced by $\mathbf{n}_i^{tm} \equiv \{n_{i1}^{tm}, n_{i2}^{tm}, \ldots, n_{iC}^{tm}\}$ jobs of classes $1, 2, \ldots, C$ respectively. We decompose the vector $\mathbf{n}_i^{tm}$ into $\mathbf{A}^{tm}$, a collection of i-independent external arrivals to the system and $\tilde{\mathbf{n}}_i^{tm}$, a set of (internal) descendants of the serviced class i job which enable the modelling of state transitions for processed jobs. For a given i, the vectors $\mathbf{A}^{tm}$ and $\tilde{\mathbf{n}}_i^{tm}$ are independent and identically distributed as (t, m) varies and $\mathbf{n}_i^{tm} = \mathbf{A}^{tm} + \tilde{\mathbf{n}}_i^{tm}$.

We now note that when $|M_t| > \sum_{j=1}^{C} N_j(t)$, where $N_j(t)$ is the number of class j jobs present and requiring service at $t \in \mathbb{N}$, there will certainly be idle machines at time t. An idle machine is deemed to be processing a class 0 job and we suppose that there are always M class 0 jobs in the system. Each machine has an allocated class 0 job; use 0_m for the one allocated to machine m, $1 \leq m \leq M$. The notation $\mathbf{n}_i^{tm}$ is extended to include the case $i = 0$, but note that $n_{i0}^{tm} = 0$ for all choices of $i \neq 0$, t and m and $\mathbf{n}_0^{tm} \equiv \{n_{01}^{tm}, n_{02}^{tm}, \ldots, n_{0C}^{tm}\} = \mathbf{A}^{tm}$.

The processing of a class i job, at time t, on any machine, earns a reward $r_i \int_t^{t+1} e^{-\alpha s} ds$ with $r_i > 0$, $1 \leq i \leq C$ and $\alpha > 0$ the discount rate. The idle class 0 earns no reward. Rewards are additive across machines and over time. The objective is to find a scheduling policy to maximise the total expected reward over an infinite time horizon. We assume that admissible scheduling policies are non-anticipative, though throughout the chapter controls may need to meet other requirements. As earlier we denote the set of admissible controls by $\mathcal{U}$ with u a typical member. We write $E = \{0, 1, 2, \ldots, C\}$ for the full set of job classes and the initial state of the system $\mathbf{N}(0) = \mathbf{k}$. The problem may be expressed as the stochastic optimisation problem

$$R^{\text{opt}}(\mathbf{k}) = \sup_{u \in \mathcal{U}} R^u(\mathbf{k}) = \sup_{u \in \mathcal{U}} \left(\sum_{j \in E} r_j x_j^u(\mathbf{k}) \right), \tag{4.5}$$

where $R^u(\mathbf{k})$ is the total expected reward earned under control u and

$$x_j^u(\mathbf{k}) = E_u \left\{ \int_0^\infty n_j(s) e^{-\alpha s} ds \;\middle|\; \mathbf{k} \right\}. \tag{4.6}$$

In (4.6), for each $t \in \mathbb{N}$, $n_j(s)$, $t \leq s \leq t+1$, is the number of class j jobs scheduled for processing at time t. The *stability* of this system under all admissible controls studied

is guaranteed by requiring that the $|E| \times |E|$ matrix $\mathbf{I} - \mathbf{n}$ be positive definite. Here $\mathbf{I}$ is the identity and $\mathbf{n}$ has $(i,j)^{\text{th}}$ entry equal to $E(n_{ij}^{tm})$, $i \in E$, $j \in E$. See Bertsimas and Niño-Mora (1996).

4.3.2 Model 2: discounted multi-armed bandit

We consider a multi-armed bandit problem on identical parallel machines modelled as in Section 2.2 but with machine availability as described at the beginning of Section 4.2. At each instant in discrete time, $t \in \mathbb{N}$, $|M_t|$ bandits are chosen for processing, one on each of the available machines. The rewards earned and evolution observed under processing of any bandits chosen are as in Section 2.2. Note in particular that we shall continue to require that the Markovian transition matrices $\mathbf{P}^b$ are irreducible and hence positive recurrent. The $B - |M_t|$ bandits not chosen for processing experience no change of state and earn no reward. Admissible controls are non-anticipative non-idling. The total expected reward earned under policy u from initial state $\mathbf{k}$ is expressed as

$$R^u(\mathbf{k}) = \sum_{i \in E} r_j x_j^u(\mathbf{k}); \tag{4.7}$$

as in (2.1) and (2.2). Our goal is the analysis of the stochastic optimisation

$$R^{\text{opt}}(\mathbf{k}) = \sup_{u \in \mathcal{U}} R^u(\mathbf{k}),$$

which is formally identical to (4.5).

4.4 Tools of analysis

For both Models 1 and 2, the objective is to find a scheduling policy to maximise the total expected discounted reward over an infinite time horizon. Recall in both models, E represents the universal set of objects to be scheduled on the available machines. As in earlier chapters, a central idea in our analyses is that in both models, by using the primal-dual approach outlined in Section 1.6, the expected rewards earned under control u may be expressed in terms of certain key quantities. These expressions follow from the structure of AG, described in Section 1.5.3, whose inputs are given by the collection $\mathbf{A} = \{\mathbf{A}^S, S \subseteq E\}$ together with the reward vector $\mathbf{r} = \{r_j, j \in E\}$. As in earlier chapters, we follow Bertsimas and Niño-Mora (1996) in their analysis of the

single machine branching bandit problem ($|M_t| = 1, t \in \mathbb{N}$), of which the multi-armed bandit problem is a special case, in defining a vector of coefficients $\mathbf{A}^S = \{A_i^S\}_{i \in E}$ for subsets of E. The general definition of $\mathbf{A}^S$ given below is the same for both Models 1 and 2. However the random variables concerned in the definition must be identified separately.

For Model 1 we define the random variables $T_i(S^c)$ as follows: fix job class $i \in E$ and subset $S \subseteq E$ with $i \in S$. Consider the discounted branching bandit model above, but in a set-up in which only a single job of class i is present at time 0, processing is provided by a single machine which is always available ($|M_t| = 1, t \in \mathbb{N}$) and the scheduling policy in operation gives jobs from classes in S^c priority over those in S. We write $T_i(S^c)$ for the first time at or after time 1 at which no S^c-jobs are present in the system. Plainly the distribution of the random variable $T_i(S^c)$ is independent of the $(S^c \to S)$ control chosen. We now note that under the condition guaranteeing stability for Model 1, given in Section 4.3.1, if $0 \in S$ then all positive moments of $T_i(S^c)$ are finite.

For Model 2, we fix $i \in E_b \subseteq E$ and subset $S \subseteq E$ with $i \in S$ and consider the discounted multi-armed bandit model described in Section 4.3.2 but in a set up in which only bandit b in state i is present at time 0 and processing is provided by a single machine which is always available ($|M_t| = 1, t \in \mathbb{N}$). We write $T_i(S^c)$ for the first time at or after time $t = 1$ at which bandit b enters S. It follows from the assumed positive recurrence of the Markov chain determined by the one-step transition matrix $\mathbf{P}^b$, that all positive moments of $T_i(S^c)$ are finite.

With the above in place, as in earlier chapters, we define $\mathbf{A}^S = \{A_i^S\}_{i \in E}$ for subsets of E as follows:

$$A_i^S = \begin{cases} \left\{1 - E\left(e^{-\alpha T_i(S^c)}\right)\right\} / \left(1 - e^{-\alpha}\right), & i \in S, \\ 0, & i \notin S. \end{cases}$$

Note that for Model 1, the definition of $\mathbf{A}^S$ above is for subsets of E containing 0.

Now, the outputs from AG, with inputs given by the matrix $\mathbf{A}^S$ and the reward vector $\mathbf{r}$, include the set of non-negative Gittins indices, G_j, $j \in E$. In line with earlier chapters we assume that the members of E are labelled $\{1, 2, \ldots, |E|\}$ such that

$$G_{|E|} \geq G_{|E|-1} \geq \ldots \geq G_2 \geq G_1$$

and define $S_j = \{j, j-1, \ldots, 1\}$ as the subset of E of cardinality j with lowest Gittins indices. In none of the results in Sections 4.4-4.6 does it matter how ties are broken between options of equal index value.

With the above in mind, we now present the following result in relation to Model 1.

Lemma 4

The idling class 0 has the smallest Gittins index.

Proof

We recall that a Gittins index scheduling policy is one which always chooses jobs with the highest index values from the collection available for processing. In the single machine case ($|M_t| = 1, t \in \mathbb{N}$) of Model 1 , by Bertsimas and Niño-Mora (1996), such a policy is optimal in the class of all non-anticipative controls. We consider a single machine problem in which at time $t = 0$, two jobs are present in the system; a single job i^* from some non-empty subset $I \subseteq E \setminus \{0\}$ and a class 0 job. We shall suppose that the index for job 0 is strictly greater than those for classes in I but less than or equal to those for classes in $E \setminus [I \cup \{0\}]$ and obtain the result stated above via a proof by contradiction.

Under the above hypothesis, a Gittins index policy will be optimal and will at time $t = 0$ schedule the class 0 job, earning no reward. Thereafter, jobs from $E \setminus I$ will be chosen for processing according to their Gittins index values and since a class 0 job is always present in the system, the job i^* will never be scheduled for processing. We write $e^{-\alpha} R$ for the reward earned from this control, since no reward is earned from the processing at time 0.

We consider now the control which at the first decision epoch selects job i^* for processing and which from time $t = 1$ onwards schedules the external arrivals during $[0, 1)$ together with job 0 and their respective descendants according to their Gittins index values. In particular, the internal descendants of job i^* at time $t = 1$ are never considered for processing. Since from time $t = 1$, this control yields outcomes which are stochastically identical to those from the Gittins index policy of the previous paragraph, it follows that the expected reward from this control is $r_{i^*} + e^{-\alpha} R$. This contradicts the optimality of the Gittins index policy and the result follows.

Since, from Lemma 4, the idling class 0 has the smallest Gittins index, we may assume, without loss of generality, that class 0 is a member of S_j for all j, $1 \leq j \leq |E|$. We then conclude that for $i \in S_j$, all positive moments of $T_i(S_j^c)$ are finite.

We now proceed, as in earlier chapters, to obtain expressions for the rewards $\{r_i,\ i \in E\}$ in terms of the quantities A_i^S, $i \in E$ and G_j, $j \in E$. Following (1.31), from the structure of AG, we obtain

$$\begin{aligned} r_i &= G_{|E|}A_i^E - \sum_{j=i}^{|E|-1}(G_{j+1} - G_j)A_i^{S_j} \\ &= G_{|E|} - \sum_{j=i}^{|E|-1}(G_{j+1} - G_j)A_i^{S_j}. \end{aligned} \tag{4.8}$$

Equation (4.8) uses the fact that, for both models, $A_i^E = 1$, $i \in E$. Lemma 5 below is a straightforward consequence of (4.5), (4.7) and (4.8) and utilises the notational shorthand

$$A^u(S, \mathbf{k}) = \sum_{i \in S} A_i^S x_i^u(\mathbf{k}), \qquad S \subseteq E.$$

Lemma 5

For both Models 1 and 2 and all initial states $\mathbf{k}$,

$$R^u(\mathbf{k}) = G_{|E|}E\left\{\int_0^\infty M(s)e^{-\alpha s}ds\right\} - \sum_{j=1}^{|E|-1}(G_{j+1} - G_j)A^u(S_j, \mathbf{k}).$$

Proof

For Models 1 and 2

$$\begin{aligned} R^u(\mathbf{k}) &= \sum_{i \in E} r_i x_i^u(\mathbf{k}) \\ &= G_{|E|}\sum_{i \in E} x_i^u(\mathbf{k}) - \sum_{j=1}^{|E|-1}(G_{j+1} - G_j)\sum_{i \in S_j} A_i^{S_j} x_i^u(\mathbf{k}) \\ &= G_{|E|}E\left\{\int_0^\infty M(s)e^{-\alpha s}ds\right\} - \sum_{j=1}^{|E|-1}(G_{j+1} - G_j)A^u(S_j, \mathbf{k}), \end{aligned} \tag{4.9}$$

where in (4.9) we utilise the fact that for both Models 1 and 2

$$\sum_{i \in E} x_i^u(\mathbf{k}) = E\left\{\int_0^\infty M(s)e^{-\alpha s}ds\right\}.$$

This concludes the proof.

Lemma 5 leads on immediately to Corollary 3, which is the key tool for evaluating proposed scheduling controls for Models 1 and 2. We firstly introduce

$$A(S, \mathbf{k}) = \inf_{u \in \mathcal{U}} A^u(S, \mathbf{k}), \quad S \subseteq E.$$

Corollary 3

(i) For both Models 1 and 2 and all initial states $\mathbf{k}$,

$$R^{\text{opt}}(\mathbf{k}) \leq G_{|E|} E\left\{\int_0^\infty M(s) e^{-\alpha s} ds\right\} - \sum_{j=1}^{|E|-1} (G_{j+1} - G_j) A(S_j, \mathbf{k});$$

(ii) For both Models 1 and 2, all admissible controls u and all initial states $\mathbf{k}$,

$$R^{\text{opt}}(\mathbf{k}) - R^u(\mathbf{k}) \leq \sum_{j=1}^{|E|-1} (G_{j+1} - G_j) \left\{A^u(S_j, \mathbf{k}) - A(S_j, \mathbf{k})\right\}.$$

The practical challenge from Corollary 3 (ii) is in deriving tight upper bounds for the associated quantities $A^u(S_j, \mathbf{k})$ and lower bounds for the $A(S_j, \mathbf{k})$ in order to obtain bounds on the degree of reward suboptimality of some given heuristic control u. In the remainder of this section we concentrate on obtaining suitable lower bounds for the quantities $A(S_j, \mathbf{k})$ for each of our models.

4.4.1 Model 1: discounted branching bandit

Before proceeding to develop effective lower bounds for $A(S_j, \mathbf{k})$ in Lemmas 6 and 7, we introduce one further piece of notation. Fix subset $S \subseteq E$ and consider a discounted branching bandit with initial state $\mathbf{k}$, on a single machine ($|M_t| = 1, t \in \mathbb{N}$). Along the lines of (2.16), we write $T_{\mathbf{k}}(S^c)$ for the first time, under a control enforcing the priority $S^c \to S$, at which no S^c-jobs are present in the system. If no S^c-jobs are present in $\mathbf{k}$ then $T_{\mathbf{k}}(S^c) = 0$. Plainly the distribution of $T_{\mathbf{k}}(S^c)$ is independent of which control from the specified class ($S^c \to S$) is utilised.

Lemma 6

When $\mathcal{U}$ is the class of all non-anticipative controls for Model 1,

$$A(S_j, \mathbf{k}) \geq E\left\{\int_{\bar{M}^{-1}(T_{\mathbf{k}}(S_j^c))}^{\infty} M(s)e^{-\alpha s}ds\right\}\left(1 - \frac{\alpha e^{2\alpha}}{2}\left[\max_{i\in S_j} E\{(T_i(S_j^c))^2\}\right]\right), \quad 1 \leq j \leq |E|, \tag{4.10}$$

for all initial states $\mathbf{k}$.

Proof

Fix policy u and denote by $\tau_{i,l}$, the time of the l^{th} occasion (for any suitable numbering of occasions) upon which the control u processes a job from class $i \in S_j$. The associated collection of independent and identically distributed random variables $\{T_{i,l}(S_j^c), l \in \mathbb{Z}^+\}$, all share the distribution of $T_i(S_j^c)$ defined for Model 1 at the beginning of Section 4.4. The aim is to find lower bounds for the quantities

$$A(S_j, \mathbf{k}) = \inf_{u\in\mathcal{U}} A^u(S_j, \mathbf{k}) = \inf_{u\in\mathcal{U}} \sum_{i\in S_j} A_i^{S_j} x_i^u(\mathbf{k}), \quad 1 \leq j \leq |E|.$$

From the definitions of the quantities involved,

$$\begin{aligned} A(S_j, \mathbf{k}) &= \inf_{u\in\mathcal{U}} \sum_{i\in S_j} A_i^{S_j} E_u\left(\sum_{l=1}^{\infty}\int_{\tau_{i,l}}^{\tau_{i,l}+1} e^{-\alpha s}ds \;\middle|\; \mathbf{k}\right) \\ &= \inf_{u\in\mathcal{U}} \sum_{i\in S_j} \frac{E\left(\int_0^{T_i(S_j^c)} e^{-\alpha s}ds\right)}{\left(\int_0^1 e^{-\alpha s}ds\right)} E_u\left(\sum_{l=1}^{\infty} e^{-\alpha\tau_{i,l}}\int_0^1 e^{-\alpha s}ds \;\middle|\; \mathbf{k}\right) \\ &= \inf_{u\in\mathcal{U}} E_u\left(\sum_{i\in S_j}\sum_{l=1}^{\infty} e^{-\alpha\tau_{i,l}}\int_0^{T_{i,l}(S_j^c)} e^{-\alpha s}ds \;\middle|\; \mathbf{k}\right). \end{aligned} \tag{4.11}$$

We now proceed to relax the minimisation in (4.11). The relaxation takes the form of a single machine scheduling problem as follows: consider the system with initial state $\mathbf{k}$ and processed by a single machine which is always available ($|M_t| = 1, t \in \mathbb{N}$). The set of non-anticipative admissible controls in the derived single machine problem is denoted by $\tilde{\mathcal{U}}$ and we use $\tilde{\tau}_{i,l}$ to denote the time of the l^{th} occasion upon which the single machine controller processes a class i job. The associated $\{T_{i,l}(S_j^c), l \in \mathbb{Z}^+\}$ are as defined above. We relax the minimisation in (4.11) by considering the stochastic

optimisation problem given by

$$\tilde{A}(S_j, \mathbf{k}) = \inf_{\tilde{u} \in \tilde{\mathcal{U}}} E_{\tilde{u}} \left[\sum_{i \in S_j} \sum_{l=1}^{\infty} \exp\{-\alpha \lceil \bar{M}^{-1}(\tilde{\tau}_{i,l}) \rceil\} \int_0^{T_{i,l}(S_j^c)} e^{-\alpha s} ds \;\middle|\; \mathbf{k} \right], \tag{4.12}$$

where $\lceil x \rceil$ is the integer part of x. To see that (4.12) is a relaxation of (4.11), note that in the objective in (4.12) rewards accruing from machine allocations at times $t = \bar{M}(s), \bar{M}(s)+1, \ldots, \bar{M}(s+1)-1$ attract discounting $e^{-\alpha s}$ for any $s \in \mathbb{N}$ and hence can be thought of as accruing at time s in problem (4.11). Since all controls in $\mathcal{U}$ for (4.11) may be mapped in an obvious way to controls in $\tilde{\mathcal{U}}$ for (4.12), it follows that for all initial states $\mathbf{k}$ and all j, $1 \leq j \leq |E|$,

$$A(S_j, \mathbf{k}) \geq \tilde{A}(S_j, \mathbf{k}). \tag{4.13}$$

The single machine problem (4.12) is easily solved. It follows from a simple pairwise interchange argument which utilises the fact that the function $e^{-\alpha s}$ is decreasing in s and that $\{\bar{M}^{-1}(s),\, s \in \mathbb{R}^+\}$ is non-decreasing on all sample paths that a minimising $\tilde{u} \in \tilde{\mathcal{U}}$ will enforce the priority $S_j^c \to S_j$. Under such a $\tilde{u}$, the first epoch at which an S_j-job is chosen in the single machine problem is $T_{\mathbf{k}}(S_j^c)$. Now suppose that the S_j-job chosen for processing at time $T_{\mathbf{k}}(S_j^c)$ is in class $i \in S_j$. Under control $\tilde{u}$, the processing of i at $T_{\mathbf{k}}(S_j^c)$ will be followed by an excursion into S_j of length $T_{i,1}(S_j^c)$. We now examine the contribution to $\tilde{A}(S_j, \mathbf{k})$ from the processing of $i \in S_j$ at $T_{\mathbf{k}}(S_j^c)$ under the objective in (4.12). In doing so we utilise the fact that $e^{-\alpha t} \geq 1 - \alpha t$, $\alpha \geq 0$, $t \geq 0$. This contribution is given by

$$\begin{aligned}
&E\left[\exp\left\{ -\alpha \lceil \bar{M}^{-1}(T_{\mathbf{k}}(S_j^c)) \rceil \right\} \int_0^{T_{i,1}(S_j^c)} e^{-\alpha s} ds \right] \\
&\qquad \geq E\left[\exp\left\{ -\alpha \lceil \bar{M}^{-1}(T_{\mathbf{k}}(S_j^c)) \rceil \right\} \int_0^{T_{i,1}(S_j^c)} \{1 - \alpha s\}\, ds \right] \\
&\qquad = E\left[\exp\left\{ -\alpha \lceil \bar{M}^{-1}(T_{\mathbf{k}}(S_j^c)) \rceil \right\} \left\{ T_{i,1}(S_j^c) - \frac{\alpha}{2} (T_{i,1}(S_j^c))^2 \right\} \right] \\
&\qquad \geq E\left(\sum_{t=T_{\mathbf{k}}(S_j^c)}^{T_{\mathbf{k}}(S_j^c)+T_{i,1}(S_j^c)-1} \int_{\lceil \bar{M}^{-1}(t) \rceil}^{\lceil \bar{M}^{-1}(t) \rceil + 1} e^{-\alpha s} ds \right) \\
&\qquad\qquad - \frac{\alpha}{2} E\left[\exp\left\{ -\alpha \lceil \bar{M}^{-1}(T_{\mathbf{k}}(S_j^c)) \rceil \right\} \right] E\left[(T_{i,1}(S_j^c))^2 \right].
\end{aligned} \tag{4.14}$$

The next contribution to $\tilde{A}(S_j, \mathbf{k})$ will occur at time $T_{\mathbf{k}}(S_j^c) + T_{i,1}(S_j^c)$ when an S_j-job is next chosen for processing. Repeating the calculation to (4.14) for successive decision epochs and aggregating, we obtain that

$$\begin{aligned}
\tilde{A}(S_j, \mathbf{k}) &\geq E\left(\sum_{t=T_{\mathbf{k}}(S_j^c)}^{\infty} \int_{\lceil \bar{M}^{-1}(t) \rceil}^{\lceil \bar{M}^{-1}(t) \rceil + 1} e^{-\alpha s} ds\right) \\
&\quad - \frac{\alpha}{2} E_{\tilde{u}}\left[\sum_{i \in S_j} \sum_{l=1}^{\infty} \exp\left\{-\alpha \lceil \bar{M}^{-1}(\tilde{\tau}_{i,l}) \rceil\right\} \middle| \mathbf{k}\right] \left[\max_{i \in S_j} E\left\{(T_i(S_j^c))^2\right\}\right] \quad (4.15) \\
&\geq E\left(\sum_{t=T_{\mathbf{k}}(S_j^c)}^{\infty} \int_{\lceil \bar{M}^{-1}(t) \rceil}^{\lceil \bar{M}^{-1}(t) \rceil + 1} e^{-\alpha s} ds\right) \left(1 - \frac{\alpha e^{\alpha}}{2} \left[\max_{i \in S_j} E\left\{(T_i(S_j^c))^2\right\}\right]\right) \quad (4.16) \\
&\geq E\left(\int_{\bar{M}^{-1}(T_{\mathbf{k}}(S_j^c))}^{\infty} M(s) e^{-\alpha s} ds\right) \left(1 - \frac{\alpha e^{2\alpha}}{2} \left[\max_{i \in S_j} E\left\{(T_i(S_j^c))^2\right\}\right]\right). \quad (4.17)
\end{aligned}$$

Note that (4.16) follows from (4.15) by bounding the second term in the latter. The bound

$$E_{\tilde{u}}\left[\sum_{i \in S_j} \sum_{l=1}^{\infty} \exp\left\{-\alpha \lceil \bar{M}^{-1}(\tilde{\tau}_{i,l}) \rceil\right\} \middle| \mathbf{k}\right] \leq e^{\alpha} E\left[\sum_{t=T_{\mathbf{k}}(S_j^c)}^{\infty} \int_{\lceil \bar{M}^{-1}(t) \rceil}^{\lceil \bar{M}^{-1}(t) \rceil + 1} e^{-\alpha s} ds\right]$$

follows by using the inclusion

$$\left\{\tilde{\tau}_{i,l};\, i \in S_j, l \in \mathbb{Z}^+\right\} \subseteq \left\{t;\, t \in \mathbb{Z}^+ \text{and } t \geq T_{\mathbf{k}}(S_j^c)\right\}.$$

Inequality (4.17) follows from (4.16) by straightforward analysis and by the definitions of the quantities involved. The result now follows from (4.13) and (4.17).

The lower bound for $A(S_j, \mathbf{k})$ presented in Lemma 6 considers the class of all non-anticipative controls. We now consider a more restrictive class of admissible controls for Model 1, namely the class of non-anticipative **local** controls. This class of policies have the property that, should a class i job be scheduled for processing at time t to some machine m, then the $\mathbf{n}_i^{tm} \equiv \{n_{i1}^{tm}, n_{i2}^{tm}, \ldots, n_{iC}^{tm}\}$ jobs of classes 1,2,...,C which replace it at time $t+1$ must also be processed on machine m. Hence jobs are processed on the same machine as all of their descendants. Since the class of local controls outlined above is more restrictive, the lower bound for $A(S_j, \mathbf{k})$, given in Lemma 7, is higher than that in Lemma 6.

Lemma 7

When $\mathcal{U}$ is the class of all non-anticipative local controls for Model 1,

$$A(S_j, \mathbf{k}) \geq E\left\{\int_{\bar{M}^{-1}(T_{\mathbf{k}}(S_j^c))}^{\infty} M(s)e^{-\alpha s}ds\right\}, \quad 1 \leq j \leq |E|, \tag{4.18}$$

for all initial states $\mathbf{k}$.

Proof

Fix policy u and denote by $\tau_{i,m,l}$, the time of the l^{th} occasion upon which the control u processes a job from class $i \in S_j$ on machine m. With each $\tau_{i,m,l}$ is associated a random variable $T_{i,m,l}(S_j^c)$ to be thought of as the number of S_j^c-descendants generated by the allocation at time $\tau_{i,m,l}$. The aim is to find lower bounds for the quantities

$$A(S_j, \mathbf{k}) = \inf_{u \in \mathcal{U}} A^u(S_j, \mathbf{k}) = \inf_{u \in \mathcal{U}} \sum_{i \in S_j} A_i^{S_j} x_i^u(\mathbf{k}), \quad 1 \leq j \leq |E|.$$

From the definitions of the quantities involved,

$$\begin{aligned}
A(S_j, \mathbf{k}) &= \inf_{u \in \mathcal{U}} \sum_{m=1}^{M} \sum_{i \in S_j} \sum_{l=1}^{\infty} A_i^{S_j} E_u \left(\int_{\tau_{i,m,l}}^{\tau_{i,m,l}+1} e^{-\alpha t} dt \;\middle|\; \mathbf{k} \right) \\
&= \inf_{u \in \mathcal{U}} \sum_{m=1}^{M} \sum_{i \in S_j} \sum_{l=1}^{\infty} \frac{E\left(\int_0^{T_{i,m,l}(S_j^c)} e^{-\alpha t} dt \right)}{\left(\int_0^1 e^{-\alpha t} dt \right)} E_u \left(e^{-\alpha \tau_{i,m,l}} \int_0^1 e^{-\alpha t} dt \;\middle|\; \mathbf{k} \right) \\
&= \inf_{u \in \mathcal{U}} \sum_{m=1}^{M} \sum_{i \in S_j} E_u \left(\sum_{l=1}^{\infty} e^{-\alpha \tau_{i,m,l}} \int_0^{T_{i,m,l}(S_j^c)} e^{-\alpha t} dt \;\middle|\; \mathbf{k} \right) \\
&= \inf_{u \in \mathcal{U}} \sum_{m=1}^{M} \sum_{i \in S_j} E_u \left(\sum_{l=1}^{\infty} \int_{\tau_{i,m,l}}^{T_{i,m,l}(S_j^c)+\tau_{i,m,l}} e^{-\alpha t} dt \;\middle|\; \mathbf{k} \right).
\end{aligned} \tag{4.19}$$

Consider now the scheduling of $i \in S_j$ at time $\tau_{i,m,l}$ on machine m. The S_j-processing generated from this allocation will be delivered under local control u during $\left[\tau_{i,m,l}, \tau_{i,m,l} + T_{i,m,l}(S_j^c)\right)$ at the earliest. Hence from (4.19), the following inequality can be inferred:

$$A(S_j, \mathbf{k}) \geq \inf_{u \in \mathcal{U}} E_u \left(\sum_{m=1}^{M} \int_0^{\infty} \xi_m^j(s) e^{-\alpha s} ds \;\middle|\; \mathbf{k} \right), \tag{4.20}$$

where for each s, $t \leq s < t+1$, with $t \in \mathbb{N}$,

$$\xi_m^j(s) = \begin{cases} 1, & \text{if machine } m \text{ processes an } S_j\text{-job or an } S_j^c\text{-descendant} \\ & \text{of a previously scheduled } S_j\text{-job at time } t; \\ 0, & \text{otherwise.} \end{cases}$$

We now proceed to bound the right-hand side of (4.20) from below by relaxing the minimisation problem. The relaxation takes the form of a single machine scheduling problem in which a reward accruing in the single machine problem at time t attracts discounting $\exp\{-\alpha\lceil M^{-1}(t)\rceil\}$, $t \in \mathbb{N}$. Hence under this derived single machine relaxation, with non-anticipative admissible controls $\tilde{\mathcal{U}}$, it follows from (4.20) that

$$A(S_j, \mathbf{k}) \geq \inf_{\tilde{u} \in \tilde{\mathcal{U}}} E_{\tilde{u}} \left[\sum_{t=0}^{\infty} \xi^j(t) \int_{\lceil \bar{M}^{-1}(t) \rceil}^{\lceil \bar{M}^{-1}(t) \rceil + 1} e^{-\alpha s} ds \;\middle|\; \mathbf{k} \right], \tag{4.21}$$

where for each $t \in \mathbb{N}$,

$$\xi^j(t) = \begin{cases} 1, & \text{if an } S_j\text{-job or an } S_j^c\text{-descendant of a previously} \\ & \text{scheduled } S_j\text{-job is processed at time } t; \\ 0, & \text{otherwise.} \end{cases}$$

Obtaining policies which achieve the infimum in the single machine problem in (4.21) with admissible controls $\tilde{\mathcal{U}}$ is straightforward. It follows from a simple pairwise interchange argument based on the fact that the function $e^{-\alpha s}$ is decreasing in s that a minimising $\tilde{u} \in \tilde{\mathcal{U}}$ will enforce the priority $S_j^c \to S_j$. For such a control

$$\xi^j(t) = 1 \quad \Longleftrightarrow \quad t \geq T_{\mathbf{k}}(S_j^c).$$

We conclude from (4.21) that

$$\begin{aligned} A(S_j, \mathbf{k}) &\geq E\left(\sum_{t=T_{\mathbf{k}}(S_j^c)}^{\infty} \int_{\lceil \bar{M}^{-1}(t) \rceil}^{\lceil \bar{M}^{-1}(t) \rceil + 1} e^{-\alpha s} ds \right) \\ &\geq E\left(\int_{M^{-1}(T_{\mathbf{k}}(S_j^c))}^{\infty} M(s) e^{-\alpha s} ds \right), \end{aligned}$$

as required.

4.4.2 Model 2: discounted multi-armed bandit

As for Model 1, to prepare for our discussion of lower bounds for $A(S_j, \mathbf{k})$ in the context of Model 2, we introduce some additional notation. Recall that members of E are

numbered such that

$$G_{|E|} \geq G_{|E|-1} \geq \ldots \geq G_2 \geq G_1$$

and the definition of $S_j = \{j, j-1, \ldots, 2, 1\}$ as the set of states with the j lowest indices. As in Section 2.3, we write

$$B_j = \{b \in B;\, E_b \cap S_j = \emptyset\}, \qquad 1 \leq j \leq |E|,$$

for the collection of bandits whose state spaces have no intersection with S_j, with

$$\mu_j = |B_j| \qquad 1 \leq j \leq |E|.$$

We note that if $\mu_j \geq M$, then there exist admissible controls which *never* schedule members of S_j. Plainly under such controls it must be true for all initial states $\mathbf{k}$ that

$$\mu_j \geq M \quad \Longrightarrow \quad A(S_j, \mathbf{k}) = 0. \tag{4.22}$$

Now fix j, $1 \leq j \leq |E|$ and consider the operation of a control processing bandits *not* in B_j only and giving priority to S_j^c over S_j. We use $T_{\mathbf{k}}(S_j^c)$ to denote the first occasion at or after time $t = 0$ at which all those bandits being processed have states in S_j.

Lemma 8

When $\mathcal{U}$ is the class of non-anticipative, non-idling controls for Model 2,

$$A(S_j, \mathbf{k}) \geq E\left[\int_{\bar{M}_{\mu_j}^{-1}(T_{\mathbf{k}}(S_j^c))}^{\infty} \{M(s) - \mu_j\}^+ e^{-\alpha s} ds\right], \qquad 1 \leq j \leq |E|,$$

for all initial states $\mathbf{k}$.

Proof

Fix policy u and denote by $\tau_{i,m,l}$, the time of the l^{th} occasion upon which under control u, machine m processes bandit b in state $i \in E_b \cap S_j$. The associated collection of independent and identically distributed random variables $\{T_{i,m,l}(S_j^c), l \in \mathbb{Z}^+\}$ all share the distribution of $T_i(S_j^c)$ defined for Model 2 at the beginning of Section 4.4. The aim is to find lower bounds for the quantities

$$A(S_j, \mathbf{k}) = \inf_{u \in \mathcal{U}} A^u(S_j, \mathbf{k}) = \inf_{u \in \mathcal{U}} \sum_{i \in S_j} A_i^{S_j} x_i^u(\mathbf{k}), \quad 1 \leq j \leq |E|.$$

Now from the definitions of the quantities involved,

$$\begin{aligned} A(S_j, \mathbf{k}) &= \inf_{u\in\mathcal{U}} \sum_{m=1}^{M} \sum_{i\in S_j} \sum_{l=1}^{\infty} A_i^{S_j} E_u \left(\int_{\tau_{i,m,l}}^{\tau_{i,m,l}+1} e^{-\alpha t} dt \;\middle|\; \mathbf{k} \right) \\ &= \inf_{u\in\mathcal{U}} \sum_{m=1}^{M} \sum_{i\in S_j} \sum_{l=1}^{\infty} \frac{E\left(\int_0^{T_{i,m,l}(S_j^c)} e^{-\alpha t} dt \right)}{\left(\int_0^1 e^{-\alpha t} dt \right)} E_u \left(e^{-\alpha \tau_{i,m,l}} \int_0^1 e^{-\alpha t} dt \;\middle|\; \mathbf{k} \right) \\ &= \inf_{u\in\mathcal{U}} \sum_{m=1}^{M} \sum_{i\in S_j} E_u \left(\sum_{l=1}^{\infty} e^{-\alpha \tau_{i,m,l}} \int_0^{T_{i,m,l}(S_j^c)} e^{-\alpha t} dt \;\middle|\; \mathbf{k} \right) \\ &= \inf_{u\in\mathcal{U}} \sum_{m=1}^{M} \sum_{i\in S_j} E_u \left(\sum_{l=1}^{\infty} \int_{\tau_{i,m,l}}^{T_{i,m,l}(S_j^c)+\tau_{i,m,l}} e^{-\alpha t} dt \;\middle|\; \mathbf{k} \right). \end{aligned} \tag{4.23}$$

Consider bandit b in state $i \in E_b \cap S_j$ chosen for processing at time $\tau_{i,m,l}$. The subsequent scheduling of bandit b until it next enters S_j will be delivered during $\left[\tau_{i,m,l},\, \tau_{i,m,l} + T_{i,m,l}(S_j^c)\right)$ at the earliest. Notice also that once a bandit has visited S_j for the first time, subsequent visits will alternate with (possibly null) excursions into S_j^c. Hence from (4.23), the following inequality can be inferred;

$$A(S_j, \mathbf{k}) \geq \inf_{u\in\mathcal{U}} E_u \left\{ \sum_{b=1}^{B} \sum_{m=1}^{M} \int_0^{\infty} \xi_{bm}^j(s)\, e^{-\alpha s} ds \;\middle|\; \mathbf{k} \right\}, \tag{4.24}$$

where for each s, $t \leq s \leq t+1$, with $t \in \mathbb{N}$,

$$\xi_{bm}^j(s) = \begin{cases} 1, & \text{if machine } m \text{ processes bandit } b \text{ at time } t, \text{ where } b \\ & \text{has paid its first visit to } S_j \text{ at some time at or before } t; \\ 0, & \text{otherwise.} \end{cases}$$

As earlier, we proceed to bound the right-hand side of (4.24) from below by means of a single machine relaxation of the stochastic optimisation problem. As in the proofs of Lemmas 6 and 7, rewards accruing in the single machine problem at time t attract discounting $\exp\left\{-\alpha\lceil \bar{M}^{-1}(t)\rceil\right\}$, $t \in \mathbb{N}$. The admissible controls $\tilde{\mathcal{U}}$ in the derived single machine problem are non-anticipative and non-idling with the additional feature that the μ_j bandits whose state spaces have no intersection with S_j may only be scheduled at certain decision epochs. Observe that by construction, these bandits never contribute to the objective in (4.24) and hence should be scheduled whenever possible; we number these bandits $1, 2, \ldots, \mu_j$. In our derived single machine relaxation, policy $\tilde{u} \in \tilde{\mathcal{U}}$ may

only schedule bandit $b \in \{1, 2, \ldots, \mu_j\}$ at times $\bar{M}(t) + \{b-1\}^+$ for those $t \in \mathbb{N}$ for which $|M_t| \geq b$. At all other epochs the single machine relaxation makes a free choice from bandits $\{\mu_j + 1, \mu_j + 2, \ldots, B\}$. Under this single machine relaxation, it follows from (4.24) that

$$A(S_j, \mathbf{k}) \geq \inf_{\tilde{u} \in \tilde{\mathcal{U}}} E_{\tilde{u}} \left[\sum_{t=0}^{\infty} \xi^j(t) \int_{\lceil \bar{M}^{-1}(t) \rceil}^{\lceil \bar{M}^{-1}(t) \rceil + 1} e^{-\alpha s} ds \;\middle|\; \mathbf{k} \right], \tag{4.25}$$

with for each $t \in \mathbb{N}$,

$$\xi^j(t) = \begin{cases} 1, & \text{if a bandit is processed at time } t, \text{ which has paid} \\ & \text{its first visit to } S_j \text{ at some time at or before } t; \\ 0, & \text{otherwise.} \end{cases}$$

This single machine problem is easily solved. Clearly since the bandits $1, 2, \ldots, \mu_j$ will never contribute to the objective on the right-hand side of (4.25) and the function $e^{-\alpha s}$ is decreasing in s, any policy $\tilde{u} \in \tilde{\mathcal{U}}$ obtaining the infimum in (4.25) should schedule these bandits at all qualifying epochs. Furthermore, it follows from a simple pairwise interchange argument that all free choices should enforce the priority $S_j^c \to S_j$. Evaluating the right-hand side of (4.25) under such a control, we have

$$A(S_j, \mathbf{k}) \geq E \left[\int_{\bar{M}_{\mu_j}^{-1}(T_{\mathbf{k}}(S_j^c))}^{\infty} \{M(s) - \mu_j\}^+ e^{-\alpha s} ds \right],$$

as required.

Recall that the practical challenge from Corollary 3 (ii) in obtaining suboptimality bounds for heuristic policies for our scheduling models is in gaining upper bounds for the quantities $A^u(S_j, \mathbf{k})$, associated with control u, and lower bounds for the $A(S_j, \mathbf{k})$. Lemmas 6 - 8 present suitable bounds for the quantities $A(S_j, \mathbf{k})$. We now proceed, in Sections 4.5 and 4.6, to propose index-based controls u and derive bounds on the resulting $A^u(S_j, \mathbf{k})$.

4.5 The evaluation of a class of index-based controls for discounted branching bandits

We design a simple non-anticipative local control for Model 1 which at time 0 divides up the jobs present in the system between the (potentially available) machines. Thereafter

each individual machine processes the resulting descendants according to an index policy whenever it is up.

Recall that for Model 1, the initial state $\mathbf{k}$ describes the collection of jobs which are present and requiring service at $t = 0$. We shall abuse notation by identifying $\mathbf{k}$ with this collection, which does not include $\{0_m\}$, $1 \leq m \leq M$, the set of M class 0 jobs, one for each machine, which we assume are always present in the system. Now, consider a partition of collection $\mathbf{k}$ given by

$$\mathbf{k} = \bigcup_{m=1}^{M} \hat{\mathbf{k}}(m)$$

where the notation indicates that the jobs in collection $\hat{\mathbf{k}}(m)$ (and their descendants) will be associated with machine m, $1 \leq m \leq M$. We use $\hat{\mathbf{k}}$ to denote this partition and develop a corresponding index based control $u_G(\hat{\mathbf{k}})$. Under policy $u_G(\hat{\mathbf{k}})$, at each time $t \in \mathbb{N}$ for which $I_m(t) = 1$, machine m processes a job chosen from $\{0_m\}$ and the descendants of the jobs in collection $\hat{\mathbf{k}}(m)$ which (is present in the system and) has maximal Gittins index. In order to obtain sub-optimality bounds for policy $u_G(\hat{\mathbf{k}})$ we shall require access to suitable upper bounds for the associated quantities $A^{u_G(\hat{\mathbf{k}})}(S_j, \mathbf{k})$, $S \subseteq E$.

Consider now the collection $\hat{\mathbf{k}}(m) \cup \{0_m\}$ evolving from time $t = 0$, where processing is provided by a single permanently available machine ($|M_t| = 1, t \in \mathbb{N}$) and the scheduling policy in operation forces the priority $S^c \to S$. In such a set-up we use $T_{\hat{\mathbf{k}}(m)}(S^c)$ for the first time at which no S^c-jobs are present. We now present Lemma 9. In the statement of the result we shall require the constants

$$K_1(S_j) = \alpha e^{\alpha} \max_{i \in S_j} \left[aE(T_i(S_j^c)) + (b-1)E\left\{ (T_i(S_j^c))^2 \right\} \right], \quad 1 \leq j \leq |E|, \tag{4.26}$$

with a and b as in (4.2).

Lemma 9

For Model 1, with the machine availability process satisfying Condition 1,

$$A^{u_G(\hat{\mathbf{k}})}(S_j, \mathbf{k}) \leq \{1 + K_1(S_j)\} \, E\left(\sum_{m=1}^{M} \left[\int_{\bar{I}_m^{-1}\left\{ T_{\hat{\mathbf{k}}(m)}(S_j^c) \right\}}^{\infty} I_m(s) e^{-\alpha s} ds \right] \right), \quad 1 \leq j \leq |E|,$$

for all initial states $\mathbf{k}$ and associated partitions $\hat{\mathbf{k}}$.

Proof

We use $A_m^{u_G(\hat{\mathbf{k}})}(S_j, \mathbf{k})$ to denote the contribution to $A^{u_G(\hat{\mathbf{k}})}(S_j, \mathbf{k})$ from the processing on machine m under control $u_G(\hat{\mathbf{k}})$ and utilise the notation in the proof of Lemma 7 to obtain that

$$A_m^{u_G(\hat{\mathbf{k}})}(S_j, \mathbf{k}) = E_{u_G(\hat{\mathbf{k}})}\left\{ \sum_{i \in S_j} \sum_{l=1}^{\infty} e^{-\alpha \tau_{i,m,l}} \int_0^{T_{i,m,l}(S_j^c)} e^{-\alpha s} ds \;\middle|\; \hat{\mathbf{k}}(m) \right\}, \quad 1 \leq m \leq M. \tag{4.27}$$

We now rewrite (4.27) and utilise the fact that $e^{\alpha t} \leq 1 + \alpha t e^{\alpha t}$, $\alpha \geq 0, t \geq 0$ to obtain that

$$\begin{aligned}
&A_m^{u_G(\hat{\mathbf{k}})}(S_j, \mathbf{k}) \\
&\quad = E_{u_G(\hat{\mathbf{k}})}\left\{ \sum_{i \in S_j} \sum_{l=1}^{\infty} e^{-\alpha \tau_{i,m,l}} \sum_{t=0}^{T_{i,m,l}(S_j^c)-1} \int_t^{t+1} e^{-\alpha s} ds \;\middle|\; \hat{\mathbf{k}}(m) \right\} \\
&\quad = E_{u_G(\hat{\mathbf{k}})}\left\{ \sum_{i \in S_j} \sum_{l=1}^{\infty} e^{-\alpha \tau_{i,m,l}} \left(\sum_{t=0}^{T_{i,m,l}(S_j^c)-1} \int_t^{t+1} \exp\left\{-\alpha \bar{I}_m^{-1}(s; \tau_{i,m,l})\right\} \right.\right. \\
&\qquad\qquad \left.\left. \times \exp\left[\alpha\left\{\bar{I}_m^{-1}(s; \tau_{i,m,l}) - s\right\}\right] ds \right) \;\middle|\; \hat{\mathbf{k}}(m) \right\} \\
&\quad \leq E_{u_G(\hat{\mathbf{k}})}\left\{ \sum_{i \in S_j} \sum_{l=1}^{\infty} e^{-\alpha \tau_{i,m,l}} \left[\sum_{t=0}^{T_{i,m,l}(S_j^c)-1} \int_t^{t+1} \exp\left\{-\alpha \bar{I}_m^{-1}(s; \tau_{i,m,l})\right\} ds \right] \;\middle|\; \hat{\mathbf{k}}(m) \right\} \\
&\qquad + \alpha E_{u_G(\hat{\mathbf{k}})}\left\{ \sum_{i \in S_j} \sum_{l=1}^{\infty} e^{-\alpha \tau_{i,m,l}} \left[\sum_{t=0}^{T_{i,m,l}(S_j^c)-1} \int_t^{t+1} \left\{\bar{I}_m^{-1}(s; \tau_{i,m,l}) - s\right\} e^{-\alpha s} ds \right] \;\middle|\; \hat{\mathbf{k}}(m) \right\}.
\end{aligned} \tag{4.28}$$

Focussing on the first term of the right-hand side of (4.28), we observe that, since under control $u_G(\hat{\mathbf{k}})$ the priority $S_j^c \to S_j$ is enforced, the first decision epoch at which machine m processes an S_j-job under control $u_G(\hat{\mathbf{k}})$ is $\bar{I}_m^{-1}\left\{T_{\hat{\mathbf{k}}(m)}(S_j^c)\right\}$. The first term gives a (discounted) measure of the availability of machine m from that point. Hence

$$\begin{aligned}
&E_{u_G(\hat{\mathbf{k}})}\left\{ \sum_{i \in S_j} \sum_{l=1}^{\infty} e^{-\alpha \tau_{i,m,l}} \left[\sum_{t=0}^{T_{i,m,l}(S_j^c)-1} \int_t^{t+1} \exp\left\{-\alpha \bar{I}_m^{-1}(s; \tau_{i,m,l})\right\} ds \right] \;\middle|\; \hat{\mathbf{k}}(m) \right\} \\
&\quad = E\left(\left[\int_{\bar{I}_m^{-1}\left\{T_{\hat{\mathbf{k}}(m)}(S_j^c)\right\}}^{\infty} I_m(s) e^{-\alpha s} ds \right] \right).
\end{aligned} \tag{4.29}$$

We now focus on the second term on the right-hand side of (4.28) and utilise the facts that $1 \geq e^{-\alpha t}$, $\alpha \geq 0$, $t \geq 0$ and that $\bar{I}_m^{-1}(s;t) - s$ is non-negative and non-decreasing in s for all $t \geq 0$ to obtain that

$$
\begin{aligned}
&E_{u_G(\hat{\mathbf{k}})}\left\{ \sum_{i \in S_j} \sum_{l=1}^{\infty} e^{-\alpha \tau_{i,m,l}} \left[\sum_{t=0}^{T_{i,m,l}(S_j^c)-1} \int_t^{t+1} \left\{ \bar{I}_m^{-1}(s;\tau_{i,m,l}) - s \right\} e^{-\alpha s} ds \right] \,\middle|\, \hat{\mathbf{k}}(m) \right\} \\
&\quad \leq E_{u_G(\hat{\mathbf{k}})}\left\{ \sum_{i \in S_j} \sum_{l=1}^{\infty} e^{-\alpha \tau_{i,m,l}} \left[\sum_{t=0}^{T_{i,m,l}(S_j^c)-1} \int_t^{t+1} \left\{ \bar{I}_m^{-1}(s;\tau_{i,m,l}) - s \right\} ds \right] \,\middle|\, \hat{\mathbf{k}}(m) \right\} \\
&\quad = E_{u_G(\hat{\mathbf{k}})}\left\{ \sum_{i \in S_j} \sum_{l=1}^{\infty} e^{-\alpha \tau_{i,m,l}} \left[\sum_{t=0}^{T_{i,m,l}(S_j^c)-1} \left\{ \bar{I}_m^{-1}\big([t+1];\tau_{i,m,l}\big) - [t+1] \right\} \right] \,\middle|\, \hat{\mathbf{k}}(m) \right\} \\
&\quad \leq E_{u_G(\hat{\mathbf{k}})}\left\{ \sum_{i \in S_j} \sum_{l=1}^{\infty} e^{-\alpha \tau_{i,m,l}} T_{i,m,l}(S_j^c) \left\{ \bar{I}_m^{-1}\big(T_{i,m,l}(S_j^c);\tau_{i,m,l}\big) - T_{i,m,l}(S_j^c) \right\} \,\middle|\, \hat{\mathbf{k}}(m) \right\}.
\end{aligned}
\tag{4.30}
$$

Hence from (4.28), (4.29) and (4.30) we deduce that

$$
\begin{aligned}
A_m^{u_G(\hat{\mathbf{k}})}(S_j, \mathbf{k}) \;\leq\; & E\left(\left[\int_{\bar{I}_m^{-1}\left\{T_{\hat{\mathbf{k}}(m)}(S_j^c)\right\}}^{\infty} I_m(s) e^{-\alpha s} ds \right] \right) \\
& + \alpha E_{u_G(\hat{\mathbf{k}})}\left[\sum_{i \in S_j} \sum_{l=1}^{\infty} e^{-\alpha \tau_{i,m,l}} T_{i,m,l}(S_j^c) \left\{ \bar{I}_m^{-1}\big(T_{i,m,l}(S_j^c);\tau_{i,m,l}\big) - T_{i,m,l}(S_j^c) \right\} \,\middle|\, \hat{\mathbf{k}}(m) \right].
\end{aligned}
\tag{4.31}
$$

We now use a simple conditioning argument and invoke Condition 1, together with the fact that

$$
E\left\{ \bar{I}_m^{-1}(s;t) \,|\, \mathcal{H}_t \right\} \leq E\left\{ \bar{J}_M^{-1}(s;t) \,|\, \mathcal{H}_t \right\}
$$

for all choices of s and t to bound the second term on the right-hand side of (4.31):

$$E_{u_G(\hat{\mathbf{k}})}\left\{\sum_{i\in S_j}\sum_{l=1}^{\infty} e^{-\alpha\tau_{i,m,l}}\, T_{i,m,l}(S_j^c)\left\{\bar{I}_m^{-1}\big(T_{i,m,l}(S_j^c);\tau_{i,m,l}\big)-T_{i,m,l}(S_j^c)\right\}\;\middle|\;\hat{\mathbf{k}}(m)\right\}$$

$$= E_{u_G(\hat{\mathbf{k}})}\left[\sum_{i\in S_j}\sum_{l=1}^{\infty} e^{-\alpha\tau_{i,m,l}}\, E\left\{T_{i,m,l}(S_j^c)\left\{\bar{I}_m^{-1}\big(T_{i,m,l}(S_j^c);\tau_{i,m,l}\big)-T_{i,m,l}(S_j^c)\right\}\middle|\mathcal{H}_{\tau_{i,m,l}}\right\}\;\middle|\;\hat{\mathbf{k}}(m)\right]$$

$$= E_{u_G(\hat{\mathbf{k}})}\left[\sum_{i\in S_j}\sum_{l=1}^{\infty} e^{-\alpha\tau_{i,m,l}}\left\{E\left(T_{i,m,l}(S_j^c)\right)E\left(\bar{I}_m^{-1}\big(T_{i,m,l}(S_j^c);\tau_{i,m,l}\big)\middle|\mathcal{H}_{\tau_{i,m,l}}\right)\right.\right.$$
$$\left.\left.-E\left[\left(T_{i,m,l}(S_j^c)\right)^2\right]\right\}\;\middle|\;\hat{\mathbf{k}}(m)\right]$$

$$\le E_{u_G(\hat{\mathbf{k}})}\left[\sum_{i\in S_j}\sum_{l=1}^{\infty} e^{-\alpha\tau_{i,m,l}}\left\{E\left(T_{i,m,l}(S_j^c)\right)E\left(\bar{J}_M^{-1}\big(T_{i,m,l}(S_j^c);\tau_{i,m,l}\big)\middle|\mathcal{H}_{\tau_{i,m,l}}\right)\right.\right.$$
$$\left.\left.-E\left[\left(T_{i,m,l}(S_j^c)\right)^2\right]\right\}\;\middle|\;\hat{\mathbf{k}}(m)\right]$$

$$\le E_{u_G(\hat{\mathbf{k}})}\left[\sum_{i\in S_j}\sum_{l=1}^{\infty} e^{-\alpha\tau_{i,m,l}}\left\{E\left(T_{i,m,l}(S_j^c)\right)\left[a+bT_{i,m,l}(S_j^c)\right]-E\left[\left(T_{i,m,l}(S_j^c)\right)^2\right]\right\}\;\middle|\;\hat{\mathbf{k}}(m)\right]$$

$$= E_{u_G(\hat{\mathbf{k}})}\left[\sum_{i\in S_j}\sum_{l=1}^{\infty} e^{-\alpha\tau_{i,m,l}}\left\{aE\left(T_{i,m,l}(S_j^c)\right)+(b-1)E\left[\left(T_{i,m,l}(S_j^c)\right)^2\right]\right\}\;\middle|\;\hat{\mathbf{k}}(m)\right]. \tag{4.32}$$

From (4.31) and (4.32) we obtain the inequality

$$A_m^{u_G(\hat{\mathbf{k}})}(S_j,\mathbf{k})$$

$$\le E\left(\left[\int_{\bar{I}_m^{-1}\left\{T_{\hat{\mathbf{k}}(m)}(S_j^c)\right\}}^{\infty} I_m(s)e^{-\alpha s}ds\right]\right)$$
$$+\alpha\max_{i\in S_j}\left[aE(T_i(S_j^c))+(b-1)E\left\{(T_i(S_j^c))^2\right\}\right]E_{u_G(\hat{\mathbf{k}})}\left\{\sum_{i\in S_j}\sum_{l=1}^{\infty} e^{-\alpha\tau_{i,m,l}}\;\middle|\;\hat{\mathbf{k}}(m)\right\} \tag{4.33}$$

$$\le \{1+K_1(S_j)\}\,E\left(\left[\int_{\bar{I}_m^{-1}\left\{T_{\hat{\mathbf{k}}(m)}(S_j^c)\right\}}^{\infty} I_m(s)e^{-\alpha s}ds\right]\right). \tag{4.34}$$

Note that (4.34) proceeds from (4.33) by bounding the second term in the latter. The bound

$$E_{u_G(\hat{\mathbf{k}})}\left(\sum_{i\in S_j}\sum_{l=1}^{\infty} e^{-\alpha\tau_{i,m,l}} \,\middle|\, \hat{\mathbf{k}}(m)\right) \leq e^{\alpha} E\left(\left[\int_{\bar{I}_m^{-1}\{T_{\hat{\mathbf{k}}(m)}(S_j^c)\}}^{\infty} I_m(s)e^{-\alpha s}ds\right]\right)$$

follows via the inclusion

$$\left\{\tau_{i,m,l}\,;\, i\in S_j, l\in\mathbb{Z}^+\right\} \subseteq \left\{t; t\in\mathbb{Z}^+, I_{m,t}=1 \text{ and } t\geq \bar{I}_m^{-1}\{T_{\hat{\mathbf{k}}(m)}(S_j^c)\}\right\}.$$

The result follows simply from inequality (4.34) and the identity

$$A^{u_G(\hat{\mathbf{k}})}(S_j,\mathbf{k}) = \sum_{m=1}^{M} A_m^{u_G(\hat{\mathbf{k}})}(S_j,\mathbf{k}). \tag{4.35}$$

This concludes the proof.

We now consider the situation of a 'machine arrival process' for Model 1. In such a situation once a machine is available, it remains permanently available thereafter. This can be described by the following condition

$$m\in M_t \quad \Longrightarrow m\in M_s \quad \forall s\in[t,\infty), 1\leq m\leq M. \tag{4.36}$$

We note that condition (4.36) is a particular instance of Example 3 in Section 4.2 and includes the important special case $|M_t| = M,\ t\in\mathbb{N}$. Under this additional condition, an improved upper bound for $A^{u_G(\hat{\mathbf{k}})}(S_j,\mathbf{k})$ is available. We tighten the bound in Lemma 9 as follows:

Lemma 10

For Model 1, with the machine availability process satisfying (4.36),

$$A^{u_G(\hat{\mathbf{k}})}(S_j,\mathbf{k}) \leq E\left(\sum_{m=1}^{M}\left[\int_{\bar{I}_m^{-1}\{T_{\hat{\mathbf{k}}(m)}(S_j^c)\}}^{\infty} I_m(s)e^{-\alpha s}ds\right]\right), \quad 1\leq j\leq |E|,$$

for all initial states $\mathbf{k}$ and associated partitions $\hat{\mathbf{k}}$.

Proof

We follow the calculations in the proof of Lemma 9 through to (4.31) and note that, under condition (4.36) we have

$$\bar{I}_m^{-1}\left(T_{i,m,l}(S_j^c);\tau_{i,m,l}\right) = T_{i,m,l}(S_j^c), \qquad 1\leq m\leq M.$$

Hence the second term on the right-hand side of (4.31) is zero under (4.36). Now use (4.35) to obtain the result.

For the goal of assessing the index-based heuristic policy $u_G(\hat{\mathbf{k}})$, we have obtained lower bounds for $A(S_j, \mathbf{k})$ in Lemmas 6 and 7 and upper bounds for $A^u(S_j, \mathbf{k})$ in Lemmas 9 and 10. We are now in a position to utilise Corollary 3 (ii) to obtain bounds on the degree of reward sub-optimality of control $u_G(\hat{\mathbf{k}})$. In the proof of Theorem 8, we shall require the constants

$$\begin{aligned} K_2(S_j) &= \frac{\alpha e^{\alpha}}{2} \max_{i \in S_j} E\left\{(T_i(S_j^c))^2\right\}, \quad 1 \leq j \leq |E|, \\ K_3(S_j, \hat{\mathbf{k}}) &= \alpha \sum_{m=1}^{M} \left(aE\{T_{\hat{\mathbf{k}}(m)}(S_j^c)\} + bE\left[\{T_{\hat{\mathbf{k}}(m)}(S_j^c)\}^2\right]\right), \quad 1 \leq j \leq |E|, \end{aligned}$$

in addition to the $K_1(S_j)$, $1 \leq j \leq |E|$, given in (4.26).

Theorem 8

(i) When $\mathcal{U}$ is the class of all non-anticipative controls for Model 1 and the machine availability process satisfies Condition 1,

$$R^{\text{opt}}(\mathbf{k}) - R^{u_G(\hat{\mathbf{k}})}(\mathbf{k}) \leq O(1); \tag{4.37}$$

(ii) When $\mathcal{U}$ is the class of all non-anticipative local controls for Model 1 and the machine availability process satisfies Condition 1,

$$R^{\text{opt}}(\mathbf{k}) - R^{u_G(\hat{\mathbf{k}})}(\mathbf{k}) \leq O(1); \tag{4.38}$$

(iii) When $\mathcal{U}$ is the class of all non-anticipative local controls for Model 1 and the machine availability control process satisfies (4.36) and Condition 1′,

$$R^{\text{opt}}(\mathbf{k}) - R^{u_G(\hat{\mathbf{k}})}(\mathbf{k}) \leq O(\alpha). \tag{4.39}$$

These results hold for all initial states $\mathbf{k}$ and associated partitions $\hat{\mathbf{k}}$.

Proof

(i). Using Lemmas 6 and 9 within Corollary 3(ii) we obtain

$$
\begin{aligned}
& R^{\mathrm{opt}}(\mathbf{k}) - R^{u_G(\hat{\mathbf{k}})}(\mathbf{k}) \\
& \leq \sum_{j=1}^{|E|-1} (G_{j+1} - G_j) \left\{ A^{u_G(\hat{\mathbf{k}})}(S_j, \mathbf{k}) - A(S_j, \mathbf{k}) \right\} \\
& \leq \sum_{j=1}^{|E|-1} (G_{j+1} - G_j) \left[\{1 + K_1(S_j)\} E \left(\sum_{m=1}^{M} \left[\int_{\bar{I}_m^{-1}\{T_{\hat{\mathbf{k}}(m)}(S_j^c)\}}^{\infty} I_m(s) e^{-\alpha s} ds \right] \right) \right] \\
& \qquad - \sum_{j=1}^{|E|-1} (G_{j+1} - G_j) \left[\{1 - K_2(S_j)\} E \left\{ \int_{\bar{M}^{-1}(T_{\hat{\mathbf{k}}}(S_j^c))}^{\infty} M(s) e^{-\alpha s} ds \right\} \right] \\
& \leq \sum_{j=1}^{|E|-1} (G_{j+1} - G_j) \left[E \left(\sum_{m=1}^{M} \left[\int_{\bar{I}_m^{-1}\{T_{\hat{\mathbf{k}}(m)}(S_j^c)\}}^{\infty} I_m(s) e^{-\alpha s} ds \right] \right) \right. \\
& \qquad \left. - E \left\{ \int_{\bar{M}^{-1}(T_{\hat{\mathbf{k}}}(S_j^c))}^{\infty} M(s) e^{-\alpha s} ds \right\} \right] \\
& \quad + \sum_{j=1}^{|E|-1} (G_{j+1} - G_j) \left[K_1(S_j) E \left(\sum_{m=1}^{M} \left[\int_{\bar{I}_m^{-1}\{T_{\hat{\mathbf{k}}(m)}(S_j^c)\}}^{\infty} I_m(s) e^{-\alpha s} ds \right] \right) \right. \\
& \qquad \left. + K_2(S_j) E \left\{ \int_{\bar{M}^{-1}(T_{\hat{\mathbf{k}}}(S_j^c))}^{\infty} M(s) e^{-\alpha s} ds \right\} \right]. \quad (4.40)
\end{aligned}
$$

Focussing on the first term on the right-hand side of (4.40), we utilise the facts that $1 \geq e^{-\alpha t} \geq 1 - \alpha t,\ \alpha \geq 0, t \geq 0$ and $M(s) = \sum_{m=1}^{M} I_m(s),\ s \in \mathbb{R}^+$, to deduce that

$$
\begin{aligned}
&E\left(\sum_{m=1}^{M}\left[\int_{\bar{I}_m^{-1}\left\{T_{\hat{\mathbf{k}}(m)}(S_j^c)\right\}}^{\infty} I_m(s)e^{-\alpha s}ds\right]\right) - E\left\{\int_{\bar{M}^{-1}(T_{\mathbf{k}}(S_j^c))}^{\infty} M(s)e^{-\alpha s}ds\right\} \\
&\quad = E\left(\sum_{m=1}^{M}\left[\int_0^{\infty} I_m(s)e^{-\alpha s}ds - \int_0^{\bar{I}_m^{-1}\left\{T_{\hat{\mathbf{k}}(m)}(S_j^c)\right\}} I_m(s)e^{-\alpha s}ds\right]\right) \\
&\qquad\qquad -E\left\{\int_0^{\infty} M(s)e^{-\alpha s}ds - \int_0^{\bar{M}^{-1}(T_{\mathbf{k}}(S_j^c))} M(s)e^{-\alpha s}ds\right\} \\
&\quad = E\left\{\int_0^{\bar{M}^{-1}(T_{\mathbf{k}}(S_j^c))} M(s)e^{-\alpha s}ds\right\} - E\left(\sum_{m=1}^{M}\left[\int_0^{\bar{I}_m^{-1}\left\{T_{\hat{\mathbf{k}}(m)}(S_j^c)\right\}} I_m(s)e^{-\alpha s}ds\right]\right) \\
&\quad \leq \left[E\left\{\int_0^{\bar{M}^{-1}(T_{\mathbf{k}}(S_j^c))} M(s)ds\right\} - E\left(\sum_{m=1}^{M}\left[\int_0^{\bar{I}_m^{-1}\left\{T_{\hat{\mathbf{k}}(m)}(S_j^c)\right\}} I_m(s)ds\right]\right)\right] \\
&\qquad\qquad +\alpha E\left(\sum_{m=1}^{M}\left[\int_0^{\bar{I}_m^{-1}\left\{T_{\hat{\mathbf{k}}(m)}(S_j^c)\right\}} sI_m(s)ds\right]\right). \qquad (4.41)
\end{aligned}
$$

Now from the definitions of the quantities involved, we may re-write the first term on the right-hand side of (4.41) as

$$
\begin{aligned}
&E\left\{\bar{M}\left[\bar{M}^{-1}(T_{\mathbf{k}}(S_j^c))\right]\right\} - E\left(\sum_{m=1}^{M}\bar{I}_m\left[\bar{I}_m^{-1}\left\{T_{\hat{\mathbf{k}}(m)}(S_j^c)\right\}\right]\right) \\
&\quad = E\left[T_{\mathbf{k}}(S_j^c)\right] - E\left(\sum_{m=1}^{M} T_{\hat{\mathbf{k}}(m)}(S_j^c)\right) \\
&\quad = 0. \qquad (4.42)
\end{aligned}
$$

We now inspect the second term on the right-hand side of (4.41) and by the application of Condition 1 we infer that

$$\alpha E\left(\sum_{m=1}^{M}\left[\int_{0}^{\bar{I}_m^{-1}\{T_{\hat{\mathbf{k}}(m)}(S_j^c)\}} s\, I_m(s)ds\right]\right)$$

$$\leq \alpha E\left(\sum_{m=1}^{M}\left[T_{\hat{\mathbf{k}}(m)}(S_j^c)\,\bar{I}_m^{-1}\{T_{\hat{\mathbf{k}}(m)}(S_j^c)\}\right]\right)$$

$$\leq \alpha E\left(\sum_{m=1}^{M}\left[T_{\hat{\mathbf{k}}(m)}(S_j^c)\,\bar{J}_M^{-1}\{T_{\hat{\mathbf{k}}(m)}(S_j^c)\}\right]\right)$$

$$\leq \alpha E\left(\sum_{m=1}^{M}\left[T_{\hat{\mathbf{k}}(m)}(S_j^c)\,\left[a+b\,T_{\hat{\mathbf{k}}(m)}(S_j^c)\right]\right]\right)$$

$$= \alpha\sum_{m=1}^{M}\left(aE\{T_{\hat{\mathbf{k}}(m)}(S_j^c)\}+bE\left[\{T_{\hat{\mathbf{k}}(m)}(S_j^c)\}^2\right]\right)$$

$$= K_3(S_j,\hat{\mathbf{k}}), \qquad 1\leq j\leq |E|-1. \tag{4.43}$$

Hence from (4.41)-(4.43) it is evident that

$$E\left(\sum_{m=1}^{M}\left[\int_{\bar{I}_m^{-1}\{T_{\hat{\mathbf{k}}(m)}(S_j^c)\}}^{\infty} I_m(s)e^{-\alpha s}ds\right]\right)-E\left\{\int_{\bar{M}^{-1}(T_{\mathbf{k}}(S_j^c))}^{\infty} M(s)e^{-\alpha s}ds\right\}$$

$$\leq K_3(S_j,\hat{\mathbf{k}}), \qquad 1\leq j\leq |E|-1. \tag{4.44}$$

We now focus on the second term on the right-hand side of (4.40). By observing that

$$\max\left\{E\Big(\sum_{m=1}^{M}\Big[\int_{\bar{I}_m^{-1}\{T_{\hat{\mathbf{k}}(m)}(S_j^c)\}}^{\infty} I_m(s)e^{-\alpha s}ds\Big]\Big),\, E\Big\{\int_{\bar{M}^{-1}(T_{\hat{\mathbf{k}}}(S_j^c))}^{\infty} M(s)e^{-\alpha s}ds\Big\}\right\}$$

$$\leq E\left\{\int_{0}^{\infty} M(s)e^{-\alpha s}ds\right\}, \tag{4.45}$$

we infer that

$$K_1(S_j)\,E\left(\sum_{m=1}^{M}\left[\int_{\bar{I}_m^{-1}\{T_{\hat{\mathbf{k}}(m)}(S_j^c)\}}^{\infty} I_m(s)e^{-\alpha s}ds\right]\right)+K_2(S_j)\,E\left\{\int_{\bar{M}^{-1}(T_{\hat{\mathbf{k}}}(S_j^c))}^{\infty} M(s)e^{-\alpha s}ds\right\}$$

$$\leq \{K_1(S_j)+K_2(S_j)\}\,E\left\{\int_{0}^{\infty} M(s)e^{-\alpha s}ds\right\}. \tag{4.46}$$

Combining (4.44) and (4.46) within (4.40) we obtain

$$R^{\text{opt}}(\mathbf{k}) - R^{u_G(\hat{\mathbf{k}})}(\mathbf{k})$$

$$\leq \sum_{j=1}^{|E|-1} (G_{j+1} - G_j) \left[\{K_1(S_j) + K_2(S_j)\} E\left\{ \int_0^\infty M(s) e^{-\alpha s} ds \right\} + K_3(S_j, \hat{\mathbf{k}}) \right]. \quad (4.47)$$

We conclude the proof by remarking that in consideration of the limit $\alpha \to 0$ all positive entries in the vector $\mathbf{A}^S$ are $O(1)$ for all subsets $S \subseteq E$. It follows that all Gittins indices G_j, $1 \leq j \leq |E|$, share this property. We observe now that the constants $K_1(S_j)$, $K_2(S_j)$ and $K_3(S_j, \hat{\mathbf{k}})$ are all $O(\alpha)$ and that

$$E\left\{ \int_0^\infty M(s) e^{-\alpha s} ds \right\} = O(1/\alpha).$$

The $O(1)$ bound in (4.37) follows.

(ii). Theorem 8 (ii) follows immediately from Theorem 8 (i) since the class of controls considered in the former is a sub-class of those in the latter. However we are still interested in the form of the bound. Utilising Lemmas 7 and 9 within Corollary 3 (ii), we obtain

$$R^{\text{opt}}(\mathbf{k}) - R^{u_G(\hat{\mathbf{k}})}(\mathbf{k})$$

$$\leq \sum_{j=1}^{|E|-1} (G_{j+1} - G_j) \left[E\left(\sum_{m=1}^{M} \left[\int_{\bar{I}_m^{-1}\{T_{\hat{\mathbf{k}}(m)}(S_j^c)\}}^\infty I_m(s) e^{-\alpha s} ds \right] \right) \right.$$

$$\left. - E\left\{ \int_{\bar{M}^{-1}(T_{\hat{\mathbf{k}}}(S_j^c))}^\infty M(s) e^{-\alpha s} ds \right\} \right]$$

$$+ \sum_{j=1}^{|E|-1} (G_{j+1} - G_j) \left[K_1(S_j) E\left(\sum_{m=1}^{M} \left[\int_{\bar{I}_m^{-1}\{T_{\hat{\mathbf{k}}(m)}(S_j^c)\}}^\infty I_m(s) e^{-\alpha s} ds \right] \right) \right. \quad (4.48)$$

$$\leq \sum_{j=1}^{|E|-1} (G_{j+1} - G_j) \left[K_1(S_j) E\left\{ \int_0^\infty M(s) e^{-\alpha s} ds \right\} + K_3(S_j, \hat{\mathbf{k}}) \right], \quad (4.49)$$

where (4.49) follows from (4.48) by combining inequalities (4.44) and (4.45). The $O(1)$ suboptimality bound in (4.38) follows by reviewing the paragraph following (4.47). We note that the $O(1)$ bound in (4.47) is higher than that in (4.49), as would be expected.

(iii). Utilising Lemmas 7 and 10 within Corollary 3 (ii) we obtain.

$$R^{\text{opt}}(\mathbf{k}) - R^{u_G(\hat{\mathbf{k}})}(\mathbf{k})$$

$$\leq \sum_{j=1}^{|E|-1} (G_{j+1} - G_j) \left[E \left(\sum_{m=1}^{M} \left[\int_{\bar{I}_m^{-1}\{T_{\hat{\mathbf{k}}(m)}(S_j^c)\}}^{\infty} I_m(s) e^{-\alpha s} ds \right] \right) \right.$$

$$\left. - E \left\{ \int_{\bar{M}^{-1}(T_{\hat{\mathbf{k}}}(S_j^c))}^{\infty} M(s) e^{-\alpha s} ds \right\} \right] \quad (4.50)$$

$$\leq \sum_{j=1}^{|E|-1} (G_{j+1} - G_j) K_3(S_j, \hat{\mathbf{k}}). \quad (4.51)$$

Inequality (4.51) follows from (4.50) using (4.44). The $O(\alpha)$ suboptimality bound in (4.39) follows.

Remark

The suboptimality bounds presented in Theorem 8 hold for all partitions $\hat{\mathbf{k}}$. To guide a choice of partition for a particular problem, we note that the only constant which depends on the partition is $K_3(S_j, \hat{\mathbf{k}})$. Inspecting (4.47), (4.49) and (4.51), we suggest a choice of $\hat{\mathbf{k}}$ which minimises

$$\sum_{j=1}^{|E|-1} \sum_{m=1}^{M} (G_{j+1} - G_j) \left(a E\{T_{\hat{\mathbf{k}}(m)}(S_j^c)\} + b E\left[\{T_{\hat{\mathbf{k}}(m)}(S_j^c)\}^2 \right] \right).$$

This minimisation can be thought of as a load balancing problem. Its solution in simple cases can be seen to encourage an equal distribution of the work in the system among the machines as would seem sensible.

In Chapter 4, as throughout the thesis, model assumptions guarantee that the expected rewards $R^{\text{opt}}(\mathbf{k})$ and $R^{u_G(\hat{\mathbf{k}})}(\mathbf{k})$ are both $O(1/\alpha)$. Theorem 8 presents bounds on the degree of reward suboptimality of policy $u_G(\hat{\mathbf{k}})$ which are $O(1)$ or $O(\alpha)$. We now take the further natural step in the analysis and explore what these results imply when the limit $\alpha \longrightarrow 0$ is taken. See Section 2.5. To facilitate the discussion, we include α in the notation $u_G(\hat{\mathbf{k}}, \alpha), R^u(\mathbf{k}, \alpha)$, $R^{\text{opt}}(\mathbf{k}, \alpha)$, $\mathbf{A}^S(\alpha)$ and $G_j(\alpha)$. It is straightforward

to establish that in all cases

$$\lim_{\alpha \to 0} A_i^S(\alpha) \equiv \bar{A}_i^S = E(T_i(S^c)), \ i \in S. \tag{4.52}$$

The model assumptions guarantee that the limits in (4.52) are finite. Hence from the structure of AG presented in Section 1.5.3, the limits

$$\lim_{\alpha \to 0} G_j(\alpha) \equiv \bar{G}_j \qquad 1 \leq j \leq |E|,$$

all exist and are finite. Recall (3.4). The limiting indices $\bar{G}_j$, $j \in E$ may be computed from AG with inputs $\bar{\mathbf{A}}$ and the reward vector $\mathbf{r}$. For Model 1, the index-based heuristic policy studied has an associated limit policy $u_{\bar{G}}(\hat{\mathbf{k}})$. This limit policy shares the structure of $u_G(\hat{\mathbf{k}}, \alpha)$, but utilises an ordering of E defined with respect to the limiting indices $\bar{G}_j$, $j \in E$. Theorem 9 concerns the asymptotic optimality of our limit policy $u_{\bar{G}}(\hat{\mathbf{k}})$.

Theorem 9 (Asymptotic optimality of limit policy $u_{\bar{G}}(\hat{\mathbf{k}})$: Model 1.)

(i) When $\mathcal{U}$ is the class of all non-anticipative controls for Model 1 and the machine availability process satisfies Condition 1, then limit policy $u_{\bar{G}}(\hat{\mathbf{k}})$ is (-1)-discount optimal and is such that

$$\lim_{\alpha \to 0} \left\{ \frac{R^{\text{opt}}(\mathbf{k}, \alpha) - R^{u_{\bar{G}}(\hat{\mathbf{k}})}(\mathbf{k}, \alpha)}{R^{\text{opt}}(\mathbf{k}, \alpha)} \right\} = 0$$

for all choices of $\hat{\mathbf{k}}$;

(ii) When $\mathcal{U}$ is the class of all non-anticipative local controls for Model 1, the machine availability process satisfies (4.36) and Condition 1′ and the limiting indices are distinct then limit policy $u_{\bar{G}}(\hat{\mathbf{k}})$ is 0-discount optimal for all choices of $\hat{\mathbf{k}}$.

Proof

(i). We proceed in a similar manner to Section 3.7. Let

$$E = \bigcup_{n=1}^{N} E_n,$$

be a partition of E such that

$$j, \ k \in E_n \Longrightarrow \bar{G}_j = \bar{G}_k$$

and each set is maximal in this respect. We use $\bar{G}^{(n)}$ to denote the common value of $\bar{G}_j$ for $j \in E_n$ and re-number the E_n such that

$$\bar{G}^{(N)} > \bar{G}^{(N-1)} > \ldots > \bar{G}^{(1)}.$$

It follows that there must exist $\bar{\alpha}$ such that for $\alpha \in (0, \bar{\alpha}]$ both $u_G(\hat{\mathbf{k}}, \alpha)$ and $u_{\bar{G}}(\hat{\mathbf{k}})$ enforce the priorities $E_N \to E_{N-1} \to \ldots \to E_1$. Write $\bar{S}_n = \bigcup_{v=1}^{n} E_v$ for the subset of E of cardinality $\sum_{v=1}^{n} |E_v|$ with lowest identifiers. Recalling equation (3.4), we note that two distinct members $j,\ k$ of E_n, $1 \leq n \leq N$ have associated Gittins indices within an $O(\alpha)$ quantity of each other. Hence using a simple elaboration of Corollary 3 (ii), we have, in an obvious notation that

$$R^{\text{opt}}(\mathbf{k}, \alpha) - R^{u_{\bar{G}}(\hat{\mathbf{k}})}(\mathbf{k}, \alpha) \leq \sum_{n=1}^{N-1} \left(\bar{G}^{(n+1)} - \bar{G}^{(n)}\right) \left\{A^{u_{\bar{G}}(\hat{\mathbf{k}})}(\bar{S}_n, \mathbf{k}, \alpha) - A(\bar{S}_n, \mathbf{k}, \alpha)\right\} + O(1). \tag{4.53}$$

Now since both $u_G(\hat{\mathbf{k}}, \alpha)$ and $u_{\bar{G}}(\hat{\mathbf{k}})$ enforce the priorities $\bar{S}_n^c \to \bar{S}_n$ for all n and $\alpha \in (0, \bar{\alpha}]$, we conclude from the earlier calculations in Section 4.5 that

$$A^{u_{\bar{G}}(\hat{\mathbf{k}})}(\bar{S}_n, \mathbf{k}, \alpha) - A(\bar{S}_n, \mathbf{k}, \alpha) \leq O(1), \qquad \alpha \in (0, \bar{\alpha}],\ 1 \leq n \leq N, \tag{4.54}$$

for all choices of partition $\hat{\mathbf{k}}$. From (4.53) and (4.54) it follows that

$$R^{\text{opt}}(\mathbf{k}, \alpha) - R^{u_{\bar{G}}(\hat{\mathbf{k}})}(\mathbf{k}, \alpha) \leq O(1), \qquad \alpha \in (0, \bar{\alpha}],$$

and the conclusions of Theorem 11 (i) follow easily.

(ii). Suppose that the limiting indices $\bar{G}_j$, $j \in E$ are all distinct, namely that

$$\bar{G}_{|E|} > \bar{G}_{|E|-1} > \ldots \bar{G}_2 > \bar{G}_1. \tag{4.55}$$

In this case it is trivial to establish the existence of $\bar{\alpha} > 0$ such that the index based policy $u_G(\hat{\mathbf{k}}, \alpha)$ and the associated limit policy $u_{\bar{G}}(\hat{\mathbf{k}})$, coincide for all $\alpha \in (0, \hat{\alpha}]$. Theorem 11 (ii) then follows easily from Theorem 8 (iii).

4.5.1 Numerical study

Further examination of the reward suboptimality bound presented for Model 1 in Theorem 8 (i) was conducted via a computational study. The computer program used to perform the study is given in Appendix H.

All systems studied had 2 job classes and arrivals into the system. The stochastic dynamics governing both external arrivals and internal descendants were determined by random irreducible matrices. For example if we denote by $\mathbf{P}^E$, the matrix governing external arrivals into the system, then P_1^E is the probability of a class 1 arrival. Similarly, if we denote we denote by $\mathbf{P}^I$, the matrix governing the internal descendants of processed jobs, then P_{12}^I is the probability of a class 2 arrival following the processing of a class 1 job. Initially the entries in each matrix, were generated from a uniform (0.1,0.9) then the matrices were normalised such that rows summed to 1. Since all entries of the resultant matrices are positive, irreducibility follows. The initial state of the system $\mathbf{N}(0) = \mathbf{k}$, was randomly generated such that the number of each job class i present at time 0 satisfied $N_i(0) \in \{1, 2, 3, 4, 5\}$, $i = 1, 2$. Machine availability was modelled by a breakdown/repair process, as in Example 2 in Section 4.2, with breakdown and repair rates $\beta_m \sim U(0.01, 0.50)$ and $\rho_m \sim U(0.50, 0.99)$, $1 \leq m \leq M$, respectively. Hence Condition 1 was satisfied. In all cases the maximum number of available machines $M = 2$. Each initial machine configuration was considered.

For each discount rate α from the set $\{1.6, 0.8, 0.4, 0.2, 0.1\}$, 500 systems were simulated with $r_i^b \sim U(2.0, 5.0)$, $i = 1, 2$. The expected optimal reward from the class of controls considered was calculated for each system using the Value Iteration Algorithm. The rewards were computed by truncating the state space of number of jobs of each class present. The truncation level $N_{\max}$, the maximum number of each job class permitted, was decided upon by running several test simulations. The limit, which varied with α, was increased until the difference between successive calculations was negligible ($\leq 10^{-7}$). The tolerance ϵ as in (2.37) was 10^{-7}.

For each partition $\hat{\mathbf{k}}$ of $\mathbf{k}$, within each system studied, the expected reward earned from policy $u_G(\hat{\mathbf{k}})$ was computed as above using a truncated state space and the Value Iteration Algorithm. As a precautionary measure in our numerical study, for each simulated system the minimum reward over all partitions was recorded and hence the maximal suboptimality when following $u_G(\hat{\mathbf{k}})$ was obtained. For each of the 500 simulated systems, the quantity $\left(R^{\text{opt}} - R_{\min}^{u_{\bar{G}}(\hat{\mathbf{k}})}\right)/R^{\text{opt}}$ was then calculated, where $R_{\min}^{u_{\bar{G}}(\hat{\mathbf{k}})}$ is the minimum reward over all partitions. Each entry in Table 4.1 gives the mean of this value over the 500 simulated systems for each discount rate α. Each initial machine configuration is considered.

Alpha	$N_{\max}$	0 machines	1 machine	2 machines
1.60	25	0.441981	0.764083	0.442135
0.80	75	0.358259	0.523394	0.358418
0.40	100	0.257826	0.326101	0.257950
0.20	200	0.161146	0.184475	0.161140
0.10	300	0.099185	0.107111	0.099259

Table 4.1: Index based heuristic: Model 1

Results

The results in Table 4.1 indicate that as we vary the discount rate α towards zero, the mean value of the quantity $\left(R^{\text{opt}} - R_{\min}^{u_{\tilde{G}}(\hat{\mathbf{k}})}\right)/R^{\text{opt}}$ tends to zero in a manner which is consistent with an $O(1)$ bound (for all partitions $\hat{\mathbf{k}}$) on the degree of reward suboptimality of policy $u_G(\hat{\mathbf{k}})$ when $\mathcal{U}$ is the class of all non-anticipative controls for Model 1 and the machine availability process satisfies Condition 1 as in Theorem 8 (i).

4.6 The evaluation of a class of index-based controls for discounted multi-armed bandits

We design a simple index-based scheduling policy for Model 2 which at time $t = 0$ partitions the collection of B bandits into M sub-collections. These sub-collections are prioritised, with the order determined by the index structure of the bandits. At each $t \in \mathbb{N}$ one bandit is chosen for scheduling from each of the $|M_t|$ sub-collections of highest priority according to a Gittins index ordering. The index heuristic in this section is similar to our block allocation policy constructed for the multi-armed bandit problem on a collection of parallel machines with varying speeds in Chapter 3.

Recall that for Model 2, the initial state of the system $\mathbf{k} \in \times_b E_b$ is the B-vector of states of the individual bandits at time $t = 0$ and that the members of $E = \bigcup_b E_b$ are numbered such that

$$G_{|E|} \geq G_{|E|-1} \geq \ldots \geq G_2 \geq G_1. \tag{4.56}$$

As in Chapter 3, we write

$$j(b) = \min\{j\,;\, j \in E_b\}$$

and then number the B bandits such that

$$j(1) > j(2) > \ldots > j(B). \tag{4.57}$$

We design a control u_G which schedules bandit $b \in \{1, 2, \ldots, M-1\}$ for processing at time $t \in \mathbb{N}$, if and only if $|M_t| \geq b$. The bandits numbered $M, M+1, \ldots, B$ are only considered for processing when $|M_t| = M$. At such decision epochs, the bandit with the largest Gittins index from among this collection, is chosen for processing. Hence according to u_G, bandit 1 with the largest guaranteed reward rate is scheduled as a first priority on the available machines, then bandit 2 and so on. Only when all M machines are up, are the bandits $\{M, M+1, \ldots, B\}$ considered for processing and then a Gittins index policy is implemented on this collection.

We recall the definition of the random variables $T_{\mathbf{k}}(S_j^c)$ for Model 2 in Section 4.4.2. We write k_m for the initial state of bandit $m, 1 \leq m \leq M$ and $\mathbf{k}_M$ for the initial state of the collection of bandits $\{M, M+1, \ldots, B\}$.

Lemma 11

For Model 2, with machine availability process satisfying Condition 1,

$$A^{u_G}(S_j, \mathbf{k}) \leq \left\{1 + K_1(S_j)\right\} E\left[\sum_{m=\mu_j+1}^{M} \left\{\int_{\bar{J}_m^{-1}(T_{k_m}(S_j^c))}^{\infty} J_m(s) e^{-\alpha s} ds\right\}\right], \quad 0 \leq \mu_j \leq M-1; \tag{4.58}$$

and

$$A^{u_G}(S_j, \mathbf{k}) = 0, \quad \mu_j \geq M, \tag{4.59}$$

for all initial states $\mathbf{k}$, where the constants $K_1(S_j)$ are given by the expression in (4.26).

Proof

We firstly notice that under u_G, no job type $j < j(M)$ will ever be chosen for processing. Since this is the collection of j for which $\mu_j \geq M$, (4.59) above follows.

Suppose now that $1 \leq m \leq M-1$ and consider the contribution to $A^{u_G}(S_j, \mathbf{k})$ from the processing of bandit m under u_G at those times $t \in \mathbb{N}$ for which $|M_t| \geq m$. Denote this contribution by $A_m^{u_G}(S_j, \mathbf{k})$. We observe from the definitions of the quantities involved that

$$j \leq j(m) - 1 \qquad \Longleftrightarrow \qquad \mu_j \geq m. \tag{4.60}$$

By the structure of u_G, both conditions in (4.60) imply that $A_m^{u_G}(S_j, \mathbf{k})$ is zero. Hence we may now suppose that $m \geq \mu_j + 1$. We denote by $\tau_{i,m,l}$, the time of the l^{th} occasion upon which, during the processing of bandit m, the state $i \in E_m \cap S_j$ is encountered. The associated collection of independent and identically distributed random variables $\left\{T_{i,m,l}(S_j^c), l \in \mathbb{Z}^+\right\}$ all share the distribution of $T_i(S_j^c)$ defined for Model 2 at the beginning of Section 4.4. From the definition of the quantities involved, we have

$$A_m^{u_G}(S_j, \mathbf{k}) = E_{u_G}\left(\sum_{i \in S_j}\sum_{l=1}^{\infty} e^{-\alpha\tau_{i,m,l}} \int_0^{T_{i,m,l}(S_j^c)} e^{-\alpha s} ds \;\middle|\; k_m\right\}. \tag{4.61}$$

We now re-write (4.61) and utilise the facts that $e^{\alpha t} \leq 1 + \alpha t e^{\alpha t}$, $\alpha \geq 0, t \geq 0$, and that $\bar{J}_m^{-1}(s,t) - s$ is non-decreasing in s for all $t \geq 0$ to obtain that

$$\begin{aligned}
&A_m^{u_G}(S_j, \mathbf{k}) \\
&= E_{u_G}\left\{\sum_{i \in S_j}\sum_{l=1}^{\infty} e^{-\alpha\tau_{i,m,l}} \sum_{t=0}^{T_{i,m,l}(S_j^c)-1} \int_t^{t+1} e^{-\alpha s} ds \;\middle|\; k_m\right\} \\
&= E_{u_G}\left\{\sum_{i \in S_j}\sum_{l=1}^{\infty} e^{-\alpha\tau_{i,m,l}} \left(\sum_{t=0}^{T_{i,m,l}(S_j^c)-1} \int_t^{t+1} \exp\left\{-\alpha \bar{J}_m^{-1}(s;\tau_{i,m,l})\right\}\right.\right. \\
&\qquad\qquad \left.\left. \times \exp\left[\alpha\left\{\bar{J}_m^{-1}(s;\tau_{i,m,l}) - s\right\}\right] ds\right) \;\middle|\; k_m\right\} \\
&\leq E_{u_G}\left\{\sum_{i \in S_j}\sum_{l=1}^{\infty} e^{-\alpha\tau_{i,m,l}} \left[\sum_{t=0}^{T_{i,m,l}(S_j^c)-1} \int_t^{t+1} \exp\left\{-\alpha \bar{J}_m^{-1}(s;\tau_{i,m,l})\right\} ds\right] \;\middle|\; k_m\right\} \\
&\quad + \alpha E_{u_G}\left\{\sum_{i \in S_j}\sum_{l=1}^{\infty} e^{-\alpha\tau_{i,m,l}} \left[\sum_{t=0}^{T_{i,m,l}(S_j^c)-1} \int_t^{t+1} \left\{\bar{J}_m^{-1}(s;\tau_{i,m,l}) - s\right\} e^{-\alpha s} ds\right] \;\middle|\; k_m\right\} \\
&\leq E\left(\left[\int_{\bar{J}_m^{-1}\left\{T_{k_m}(S_j^c)\right\}}^{\infty} J_m(s) e^{-\alpha s} ds\right]\right) \\
&\quad + \alpha E_{u_G}\left\{\sum_{i \in S_j}\sum_{l=1}^{\infty} e^{-\alpha\tau_{i,m,l}} T_{i,m,l}(S_j^c)\left\{\bar{J}_m^{-1}(T_{i,m,l}(S_j^c);\tau_{i,m,l}) - T_{i,m,l}(S_j^c)\right\} \;\middle|\; k_m\right\}.
\end{aligned} \tag{4.62}$$

The details are similar to those which yield (4.31) in the proof of Lemma 9. We now use a simple conditioning argument and invoke Condition 1, together with the fact that

$$E\left\{\bar{J}_m^{-1}(s;t) \,|\, \mathcal{H}_t\right\} \leq E\left\{\bar{J}_M^{-1}(s;t) \,|\, \mathcal{H}_t\right\}$$

for all choices of s and t to bound the second term on the right-hand side of (4.62) as follows:

$$
\begin{aligned}
&E_{u_G}\left\{\sum_{i\in S_j}\sum_{l=1}^{\infty} e^{-\alpha\tau_{i,m,l}}\, T_{i,m,l}(S_j^c)\left\{\bar{J}_m^{-1}(T_{i,m,l}(S_j^c);\tau_{i,m,l}) - T_{i,m,l}(S_j^c)\right\} \;\middle|\; k_m\right\} \\
&= E_{u_G}\left[\sum_{i\in S_j}\sum_{l=1}^{\infty} e^{-\alpha\tau_{i,m,l}}\, E\left\{T_{i,m,l}(S_j^c)\left\{\bar{J}_m^{-1}(T_{i,m,l}(S_j^c);\tau_{i,m,l}) - T_{i,m,l}(S_j^c)\right\}\middle|\mathcal{H}_{\tau_{i,m,l}}\right\} \;\middle|\; k_m\right] \\
&= E_{u_G}\left[\sum_{i\in S_j}\sum_{l=1}^{\infty} e^{-\alpha\tau_{i,m,l}}\left\{E\left(T_{i,m,l}(S_j^c)\right) E\left(\bar{J}_m^{-1}(T_{i,m,l}(S_j^c);\tau_{i,m,l})\middle|\mathcal{H}_{\tau_{i,m,l}}\right)\right.\right. \\
&\qquad\qquad \left.\left. -E\left[\left(T_{i,m,l}(S_j^c)\right)^2\right]\right\} \;\middle|\; k_m\right] \\
&\le E_{u_G}\left[\sum_{i\in S_j}\sum_{l=1}^{\infty} e^{-\alpha\tau_{i,m,l}}\left\{E\left(T_{i,m,l}(S_j^c)\right) E\left(\bar{J}_M^{-1}(T_{i,m,l}(S_j^c);\tau_{i,m,l})\middle|\mathcal{H}_{\tau_{i,m,l}}\right)\right.\right. \\
&\qquad\qquad \left.\left. -E\left[\left(T_{i,m,l}(S_j^c)\right)^2\right]\right\} \;\middle|\; k_m\right] \\
&\le E_{u_G}\left[\sum_{i\in S_j}\sum_{l=1}^{\infty} e^{-\alpha\tau_{i,m,l}}\left\{E\left(T_{i,m,l}(S_j^c)\right)\left[a+bT_{i,m,l}(S_j^c)\right] - E\left[\left(T_{i,m,l}(S_j^c)\right)^2\right]\right\} \;\middle|\; k_m\right] \\
&= E_{u_G}\left[\sum_{i\in S_j}\sum_{l=1}^{\infty} e^{-\alpha\tau_{i,m,l}}\left\{aE\left(T_{i,m,l}(S_j^c)\right) + (b-1)E\left[\left(T_{i,m,l}(S_j^c)\right)^2\right]\right\} \;\middle|\; k_m\right]. \qquad (4.63)
\end{aligned}
$$

From (4.62) and (4.63) we obtain the inequality

$$
\begin{aligned}
&A_m^{u_G}(S_j,\mathbf{k}) \\
&\le E\left(\left[\int_{\bar{J}_m^{-1}\{T_{k_m}(S_j^c)\}}^{\infty} J_m(s)e^{-\alpha s}ds\right]\right) \\
&\qquad +\alpha\max_{i\in S_j}\left[aE(T_i(S_j^c)) + (b-1)E\left\{(T_i(S_j^c))^2\right\}\right] E_{u_G}\left\{\sum_{i\in S_j}\sum_{l=1}^{\infty} e^{-\alpha\tau_{i,m,l}} \;\middle|\; k_m\right\} \qquad (4.64) \\
&\le \{1+K_1(S_j)\}\, E\left(\left[\int_{\bar{J}_m^{-1}\{T_{k_m}(S_j^c)\}}^{\infty} J_m(s)e^{-\alpha s}ds\right]\right). \qquad (4.65)
\end{aligned}
$$

Note that (4.65) proceeds from (4.64) by bounding the second term in the latter. The

bound

$$E_{u_G}\left(\sum_{i\in S_j}\sum_{l=1}^{\infty} e^{-\alpha\tau_{i,m,l}} \;\middle|\; k_m\right) \leq e^{\alpha} E\left(\left[\int_{\bar{J}_m^{-1}\{T_{k_m}(S_j^c)\}}^{\infty} J_m(s)e^{-\alpha s}ds\right]\right)$$

follows via the inclusion

$$\left\{\tau_{i,m,l};\, i\in S_j, l\in\mathbb{Z}^+\right\} \subseteq \left\{t;\, t\in\mathbb{Z}^+,\, J_{m,t}=1 \text{ and } t\geq \bar{J}_m^{-1}[T_{k_m}(S_j^c)]\right\}.$$

We now write $A_M^{u_G}(S_j,\mathbf{k})$ for the contribution to $A^{u_G}(S_j,\mathbf{k})$ from the processing of the bandits $M, M+1,\ldots,B$ at those epochs for which $|M_t|=M$. By a similar account to the above, we have that

$$A_M^{u_G}(S_j,\mathbf{k}) \leq \{1+K_1(S_j)\}\, E\left(\left[\int_{\bar{J}_M^{-1}\{T_{\mathbf{k}_M}(S_j^c)\}}^{\infty} J_M(s)e^{-\alpha s}ds\right]\right). \tag{4.66}$$

Finally, observing that when $0\leq\mu_j\leq M-1$,

$$A^{u_G}(S_j,\mathbf{k}) = \sum_{m=\mu_j+1}^{M} A_m^{u_G}(S_j,\mathbf{k}), \tag{4.67}$$

(4.58) now follows from (4.65)-(4.67). This concludes the proof.

We now consider the situation, for Model 2, in which $|M_t|$ is non-decreasing in t. Under this additional condition the upper bound in Lemma 11 may be tightened. The proof of Lemma 12 modifies that of Lemma 11 in much the same way that the proof of Lemma 10 modifies that of Lemma 9.

Lemma 12

For Model 2, with $|M_t|$ non-decreasing in t, we have that

$$A^{u_G}(S_j,\mathbf{k}) \leq E\left(\sum_{m=\mu_j+1}^{M}\left[\int_{\bar{J}_m^{-1}\{T_{k_m}(S_j^c)\}}^{\infty} J_m(s)e^{-\alpha s}ds\right]\right), \quad 0\leq\mu_j\leq M-1;$$

and

$$A^{u_G}(S_j,\mathbf{k}) = 0, \quad \mu_j\geq M,$$

for all initial states $\mathbf{k}$.

Proof

Following the calculation of the proof of Lemma 11 through to (4.62), we note that with $|M_t|$ non-decreasing in t

$$\bar{J}_m^{-1}(T_{i,m,l}(S_j^c); \tau_{i,m,l}) = T_{i,m,l}(S_j^c), \qquad 1 \leq m \leq M.$$

Hence the second term on the right-hand side of (4.62) is zero with $|M_t|$ non-decreasing in t. Now use (4.67) to obtain the result.

We now use Corollary 3 (ii) to bound the degree of reward suboptimality in the index-based policy u_G. Theorem 10 follows from Corollary 3 (ii) together with Lemmas 8, 11 and 12 through an analysis similar to that in the proof of Theorem 8. In the proof of Theorem 10 we shall require the constants

$$K_4(S_j) = \alpha \sum_{m=\mu_j+1}^{M} \left(aE\{T_{k_m}(S_j^c)\} + bE[\{T_{k_m}(S_j^c)\}^2] \right)$$

for the range $0 \leq \mu_j \leq M-1$ with a and b as in (4.3), together with the constants $K_1(S_j)$, $1 \leq j \leq |E|$, given in (4.26).

Theorem 10

(i) When $\mathcal{U}$ is the class of all non-anticipative, non-idling controls for Model 2, with the machine availability process satisfying Condition 1,

$$R^{\text{opt}}(\mathbf{k}) - R^{u_G}(\mathbf{k}) \leq O(1); \tag{4.68}$$

(ii) When $\mathcal{U}$ is the class of all non-anticipative, non-idling controls for Model 2, with $|M_t|$ non-decreasing in t almost surely,

$$R^{\text{opt}}(\mathbf{k}) - R^{u_G}(\mathbf{k}) \leq O(\alpha). \tag{4.69}$$

These results hold for all initial states $\mathbf{k}$.

Proof

(i). Utilising Lemmas 8 and 11 within Corollary 3 (ii) we obtain

$$R^{\text{opt}}(\mathbf{k}) - R^{u_G}(\mathbf{k})$$

$$\leq \sum_{j=1}^{|E|-1} (G_{j+1} - G_j) \Bigg[E \left(\sum_{m=\mu_j+1}^{M} \left[\int_{\bar{J}_m^{-1}\{T_{k_m}(S_j^c)\}}^{\infty} J_m(s) e^{-\alpha s} ds \right] \right)$$

$$- E \left\{ \int_{\bar{M}_{\mu_j}^{-1}(T_{\mathbf{k}}(S_j^c))}^{\infty} \{M(s) - \mu_j\}^+ e^{-\alpha s} ds \right\} \Bigg]$$

$$+ \sum_{j=1}^{|E|-1} (G_{j+1} - G_j) \left[K_1(S_j) E \left(\sum_{m=\mu_j+1}^{M} \left[\int_{\bar{J}_m^{-1}\{T_{k_m}(S_j^c)\}}^{\infty} J_m(s) e^{-\alpha s} ds \right] \right) \right]. \quad (4.70)$$

Focussing on the first term on the right-hand side of (4.70), we utilise the facts that $1 \geq e^{-\alpha t} \geq 1 - \alpha t,\ \alpha \geq 0, t \geq 0$ to deduce that

$$E \left(\sum_{m=\mu_j+1}^{M} \left[\int_{\bar{J}_m^{-1}\{T_{k_m}(S_j^c)\}}^{\infty} J_m(s) e^{-\alpha s} ds \right] \right) - E \left\{ \int_{\bar{M}_{\mu_j}^{-1}(T_{\mathbf{k}}(S_j^c))}^{\infty} \{M(s) - \mu_j\}^+ e^{-\alpha s} ds \right\}$$

$$= E \left(\sum_{m=\mu_j+1}^{M} \left[\int_0^{\infty} J_m(s) e^{-\alpha s} ds - \int_0^{\bar{J}_m^{-1}\{T_{k_m}(S_j^c)\}} J_m(s) e^{-\alpha s} ds \right] \right)$$

$$- E \left\{ \int_0^{\infty} \{M(s) - \mu_j\}^+ e^{-\alpha s} ds - \int_0^{\bar{M}_{\mu_j}^{-1}(T_{\mathbf{k}}(S_j^c))} \{M(s) - \mu_j\}^+ e^{-\alpha s} ds \right\}$$

$$= E \left\{ \int_0^{\bar{M}_{\mu_j}^{-1}(T_{\mathbf{k}}(S_j^c))} \{M(s) - \mu_j\}^+ e^{-\alpha s} ds \right\} - E \left(\sum_{m=\mu_j+1}^{M} \left[\int_0^{\bar{J}_m^{-1}\{T_{k_m}(S_j^c)\}} J_m(s) e^{-\alpha s} ds \right] \right)$$

$$\leq E \left\{ \int_0^{\bar{M}_{\mu_j}^{-1}(T_{\mathbf{k}}(S_j^c))} \{M(s) - \mu_j\}^+ ds \right\} - E \left(\sum_{m=\mu_j+1}^{M} \left[\int_0^{\bar{J}_m^{-1}\{T_{k_m}(S_j^c)\}} J_m(s) ds \right] \right)$$

$$+ \alpha E \left(\sum_{m=\mu_j+1}^{M} \left[\int_0^{\bar{J}_m^{-1}\{T_{k_m}(S_j^c)\}} s\, J_m(s) ds \right] \right)$$

$$\leq E \left[\left\{ T_{\mathbf{k}}(S_j^c) - \sum_{m=\mu_j+1}^{M} T_{k_m}(S_j^c) \right\} \right] + \alpha E \left(\sum_{m=\mu_j+1}^{M} \left[T_{k_m}(S_j^c)\, \bar{J}_m^{-1}\{T_{k_m}(S_j^c)\} \right] \right) \quad (4.71)$$

$$\leq K_4(S_j), \qquad 0 \leq \mu_j \leq M-1, \quad (4.72)$$

by application of Condition 1 to the second term of (4.71). Note that the first term is zero. The details are similar to those which yield (4.44) in the proof of Theorem 8 (i). We observe that

$$E\left(\sum_{m=\mu_j+1}^{M}\left[\int_{\bar{J}_m^{-1}\{T_{km}(S_j^c)\}}^{\infty} J_m(s)e^{-\alpha s}ds\right]\right) \leq E\left\{\int_0^{\infty} M(s)e^{-\alpha s}ds\right\}. \tag{4.73}$$

Combining (4.72) and (4.73) within (4.70), we obtain that

$$R^{\text{opt}}(\mathbf{k}) - R^{u_G}(\mathbf{k}) \leq \sum_{j=1}^{|E|-1}(G_{j+1}-G_j)\left[K_1(S_j)E\left\{\int_0^{\infty} M(s)e^{-\alpha s}ds\right\} + K_4(S_j)\right].$$

The $O(1)$ bound in (4.68) follows by observing that $K_4(S_j)$ is $O(\alpha)$ and reviewing the paragraph following (4.47).

(ii). Utilising Lemmas 8 and 12 within Corollary 3 (ii) we obtain

$$\begin{aligned} & R^{\text{opt}}(\mathbf{k}) - R^{u_G}(\mathbf{k}) \\ & \leq \sum_{j=1}^{|E|-1}(G_{j+1}-G_j)\left[E\left(\sum_{m=\mu_j+1}^{M}\left[\int_{\bar{J}_m^{-1}\{T_{km}(S_j^c)\}}^{\infty} J_m(s)e^{-\alpha s}ds\right]\right)\right. \\ & \qquad \left. -E\left\{\int_{\bar{M}_{\mu_j}^{-1}(T_{\mathbf{k}}(S_j^c))}^{\infty}\{M(s)-\mu_j\}^{+}e^{-\alpha s}ds\right\}\right] \qquad (4.74) \\ & \leq \sum_{j=1}^{|E|-1}(G_{j+1}-G_j)K_4(S_j). \qquad (4.75) \end{aligned}$$

Inequality (4.75) follows from (4.74) using (4.72). The result follows.

Theorem 10 presents bounds on the degree of reward sub-optimality of index based controls which are proven to be $O(1)$ or $O(\alpha)$. As in Section 4.5 we explore the implications of these results as $\alpha \to 0$. We include α in the notations $u_G(\alpha)$ and $G_j(\alpha)$. In Theorems 11 and 12 various forms of asymptotic optimality are claimed for the limit policy $u_{\bar{G}}$ which utilises an ordering on E given by the limiting indices $\bar{G}_j$, $j \in E$ which are defined in the preamble to Theorem 9.

The proof of Theorem 11 is along similar lines to that of Theorem 9 and is omitted.

Theorem 11 (Asymptotic optimality of limit policy $u_{\bar{G}}$: Model 2.)

(i) When $\mathcal{U}$ is the class of all non-anticipative, non-idling controls for Model 2 and the machine availability process satisfies Condition 1, then limit policy $u_{\bar{G}}$ is (-1)-discount optimal and is such that

$$\lim_{\alpha \to 0} \left\{ \frac{R^{\text{opt}}(\mathbf{k}, \alpha) - R^{u_{\bar{G}}}(\mathbf{k}, \alpha)}{R^{\text{opt}}(\mathbf{k}, \alpha)} \right\} = 0;$$

for all choices of $\hat{\mathbf{k}}$;

(ii) When $\mathcal{U}$ is the class of all non-anticipative, non-idling controls for Model 2, with $|M_t|$ non-decreasing in t almost surely and the limiting indices are distinct, then limit policy $u_{\bar{G}}$ is 0-discount optimal.

We continue our consideration of Model 2 by imposing the additional condition that the process $\{|M_t| t \in \mathbb{N}\}$ be an irreducible Markov chain, as in Example 1. Under this condition, the multi-armed bandit of Model 2 can then be viewed as a finite state, finite action Markov decision process and we can utilise the classical theory of Blackwell (1962), Veinott (1966) and Denardo and Miller (1968), outlined in Section 2.5. We firstly observe that straightforward analysis yields the existence of a sequence $\{\alpha_n, n \in \mathbb{N}\}$ with $\lim_{n \to \infty} \alpha_n = 0$ together with a numbering of members of E for which

(i) $G_{|E|}(\alpha_n) \geq G_{|E|-1}(\alpha_n) \geq \ldots \geq G_2(\alpha_n) \geq G_1(\alpha_n)$, $n \in \mathbb{N}$;

(ii) $\bar{G}_{|E|} \geq \bar{G}_{|E|-1} \geq \ldots \geq \bar{G}_2 \geq \bar{G}_1$

in all cases. The limit policy discussed in Theorems 12 is constructed via the index ordering $|E| \to |E| - 1 \to \ldots \to 2 \to 1$ in (i) and (ii) above. The result follow from Theorem 10 (i).

Theorem 12 (Asymptotic optimality of limit policy $u_{\bar{G}}$: Markovian Model 2.)

When $\mathcal{U}$ is the class of all non-anticipative, non-idling controls for Model 2 and process $\{|M_t| t \in \mathbb{N}\}$ is an irreducible Markov chain, then limit policy $u_{\bar{G}}$ is average reward optimal.

4.6.1 Numerical study

Further examination of the reward suboptimality bounds presented for Model 2 in Theorem 10 was conducted via a computational study. The computer program used to perform the study is given in Appendix I.

The systems simulated were four-armed bandit problems, with each bandit having four states. Random irreducible transition matrices, reward vectors and an initial state $\mathbf{k}$ were generated for each problem. The machine availability process was determined by an irreducible transition matrix, as in Example 1 in Section 4.2 and so Condition 1 was always satisfied. The method used to generate all the transition matrices was as described in Section 3.4.1. The maximum number of available machines was $M = 2$. Each initial machine configuration was considered.

For each α from the set $\{0.1, 0.05, 0.025, 0.01, 0.005, 0.0025, 0.001\}$, 500 four-armed bandit problems were simulated with $r_i^b \sim U(2.0, 5.0)$, $i = 1, 2$. The Value Iteration Algorithm was used to calculate the expected reward earned when following policy u_G and also to calculate the optimal expected reward from the class of non-anticipative, non-idling controls. The tolerance ϵ as in (2.37) was 10^{-7}. The results of the study are given in the following tables. Tables 4.2 and 4.3 contain the summary statistics (as outlined in Section 3.4.1) for the simulated suboptimalities $R^{\text{opt}}(\mathbf{k}) - R^{u_G}(\mathbf{k})$ for policy u_G when $\mathcal{U}$ is the class of policies considered in Theorem 10 (i) and Theorem 10 (ii) respectively.

Alpha	Minimum	Median	Maximum	Mean	Std Dev	Percent
0.1000	0.000000	0.104872	2.657979	0.150073	0.234651	19.20000
0.0500	0.000000	0.067826	1.186225	0.081162	0.140609	38.40000
0.0250	0.000000	0.057549	0.594180	0.041592	0.094716	61.00000
0.0100	0.000000	0.035684	1.057889	0.017228	0.074540	82.40000
0.0050	0.000000	0.017479	0.760552	0.007607	0.045453	88.60000
0.0025	0.000000	0.004668	0.459982	0.003340	0.030812	95.00000
0.0010	0.000000	0.026701	0.347937	0.001886	0.023028	98.40000

Table 4.2: Index based heuristic: Model 2

Alpha	Minimum	Median	Maximum	Mean	Std Dev	Percent
0.1000	0.000000	0.040832	0.598789	0.030792	0.084385	67.80000
0.0500	0.000000	0.033434	0.508119	0.012303	0.044158	82.20000
0.0250	0.000000	0.019472	0.403589	0.006114	0.032890	89.40000
0.0100	0.000000	0.003831	0.067882	0.000572	0.004675	96.40000
0.0050	0.000000	0.004058	0.011425	0.000084	0.000825	98.40000
0.0025	0.000000	0.000203	0.001569	0.000006	0.000084	99.00000
0.0010	0.000000	0.001187	0.001187	0.000002	0.000053	99.80000

Table 4.3: Index based heuristic: Model 2: $|M_t|$ non-decreasing

Results

We observe from Table 4.2 that the variation in the summary statistics (excluding percentage of zero suboptimalities) associated with a change of discount rate is indicative of an $O(\alpha)$ bound on the degree of reward suboptimality of u_G, when $\mathcal{U}$ is the class of all non-anticipative, non-idling controls for Model 2 with the machine availability process satisfying Condition 1. These results suggest that there may be a degree of conservatism in the suboptimality bound in Theorem 10 (i).

Table 4.3 contains the summary statistics relating to the simulated suboptimalities for u_G when $\mathcal{U}$ is the class of all non-anticipative, non-idling controls for Model 2, with $|M_t|$ non-decreasing in t. These results indicate that the order of magnitude of the bound on the degree of reward suboptimality of u_G is of an order higher than order alpha. This suggests that the bound in Theorem 10 (ii) may also be conservative to a degree.

Interestingly, the numerical results in Tables 4.2 and 4.3 reflect a difference in the order of magnitude of the bounds that has emerged from the primal-dual theory. The conservatism in the theoretical results suggested by the computations may be explained by reasons similar to those outlined in Section 3.5.1, namely that the derived bounds on the rewards are not sufficiently tight or that the numerical studies focus on sub-classes of problems for which the suboptimality bounds are tighter than for the more general classes of system considered in the theory.

Chapter 5

Conclusions

5.1 Summary

We have presented a discussion of bandit problems on parallel machines which extends the analysis by Glazebrook and Wilkinson (2000) of the discounted multi-armed bandit problem on a collection of identical machines in parallel. The more general models considered were models in which bandits compete for processing by machines of varying speed and where the number of available machines is a stochastic process.

The ideas which guided our analysis of these complex stochastic scheduling problems emerged from the so-called achievable region approach to stochastic optimisation. We followed Glazebrook and Wilkinson (2000) by utilising and developing elements of the account given by Bertsimas and Niño-Mora (1996), of Gittins indexation for the branching bandit model on a single machine cast in the achievable region framework. By exploiting the primal-dual approach within achievable region methodology, we were able to design simple heuristic index-based policies for extensions of this model to the parallel machine environment.

The analysis produced performance guarantees for a range of policies based on Gittins indices. From these guarantees, various forms of asymptotic optimality have been established for the policies concerned. The tightest sub-optimality bounds for the index-based heuristics of interest available from the primal dual theory have been compared numerically with actual suboptimalities obtained from dynamic programming. These comparisons have allowed insights concerning the degree of conservatism in the theoretical results.

5.2 Methods and Tools

Bertsimas and Niño-Mora (1996) established that in the single machine case, branching bandit processes satisfy generalised conservation laws (GCLs) as defined in Section 1.5.2. As such, optimal policies are priority index policies derived from the so-called adaptive greedy algorithm AG, given in Section 1.5.3. In the parallel machine environment however, GCLs and hence the conditions sufficient to establish the optimality of Gittins index policies, just fail. Glazebrook and Garbe (1999) explained how, by exploiting the primal-dual approach (Section 1.6) within achievable region methodology, Gittins index policies can be analysed for systems in which GCLs fail narrowly. Glazebrook and Wilkinson (2000) developed the tools used by Glazebrook and Garbe (1999) to facilitate their analysis of index based polices for the parallel machine version of the discounted multi-armed bandit problem. We further developed these tools in our investigation of bandit problems on (sometimes) non-identical parallel machines.

Broadly speaking we established performance guarantees for the policies investigated by obtaining upper bounds for the optimal rewards and deriving expressions (or lower bounds) for the expected discounted rewards earned when implementing our proposed heuristics. The tools used provided effective analyses of the problems considered. The conclusions drawn from the theoretical results were, in the majority of cases supported by the results of the numerical studies. This suggests that in these cases, the bounds obtained from the primal-dual theory are, in terms of order of magnitude, the best results attainable.

In certain cases the numerical results suggested a degree of conservatism in the suboptimality bounds obtained from the theory. There may be a number of reasons for this, the most straightforward being that the derived bounds on the rewards are not sufficiently tight. A further explanation may be that the numerical studies are (inadvertently) confined to sub-classes of problems for which the suboptimality bounds are tighter than for the more general classes of systems analysed in the theory. It may be the case that though the theory makes very general claims about wide classes of systems, by focussing on more specific models in the numerical investigations, the claims of the theory appear conservative.

5.3 Index-based scheduling policies

We have analysed a number of complex stochastic scheduling problems and produced performance guarantees for a range of heuristic policies based on Gittins indices. The block allocation policy derived in Chapter 3 for the multi-armed bandit problem on a collection of machines with varying speeds and the controls designed in Chapter 4 for the branching bandit and multi-armed bandit problems on parallel machines with stochastic availability, share certain appealing characteristics. The policies considered partition the jobs between machines at time $t = 0$. For certain systems this initial once for all allocation of bandits to machines must take account of the index structure of the bandits present in the system. Following time 0, individual machines process their allocated workload according to a Gittins index policy.

The index-based heuristics proposed in Chapters 3 and 4 are remarkably simple. A single irrevocable decision is made at time zero concerning the processing rights of each bandit. The simplicity of the heuristics is highlighted when considering the various classes of admissible controls within which the performance of the proposed policies are investigated. Despite the structure of the evaluated controls delimiting the freedom of action considerably, even for complex scenarios, our policies perform extremely well.

5.4 Further work

In this section we discuss some unresolved issues and areas of possible development emerging from our analyses in Chapters 3 and 4. We begin by recalling our investigation of the class of general policies for multi-armed bandit problems on parallel machines with varying speeds, in Section 3.5. One question left unanswered in this section was whether Theorems 5 and 6 continue to hold for the situation where the machine speeds are positive irrational numbers. From a practical point of view, consideration of irrational machine speeds is immaterial, though for mathematical completeness, confirmation of this result would be desirable. Initial attempts indicate that establishing this result is not straightforward due to the complex decision structure in such cases.

Throughout the thesis, we encounter instances in which our numerical results imply a degree of conservatism in the associated theoretical suboptimality bounds. For example, computations of Section 3.5.1 regarding the performance of our block allocation

policy within the class of general policies are indicative of an $O(\alpha)$ bound on the degree of reward suboptimality whereas the associated theoretical bound in Theorem 5 (ii) is $O(1)$. As mentioned, there may be several reasons for such results. One simple explanation is that the bounds on the rewards derived from the theory are not sufficiently tight, though initial efforts to tighten the lower bound in Lemma 3 in order to improve the bound on the degree of reward suboptimality in Theorem 5 (ii) have proven ineffective. Alternatively, it may be that by focussing on specific models in the numerical studies, the very general claims made about the wide classes of systems in the theory appear more conservative. As has been stated, general computational problems occur with the Value Iteration Algorithm when the state space $\mathbf{X}$ is large and also as the discount rate $\alpha \rightarrow 0$, therefore we are limited in the size of systems we can analyse effectively. Further investigation, both numerical and theoretical, into the cases for which our computations suggest conservatism in the theoretical bounds may resolve some of these issues.

In the discussion in Section 3.6 of the performance of the Gittins index policy when implemented in the parallel machine environment with machines operating at different speeds, we describe conditions under which our block allocation policy outperforms the Gittins index policy. In light of the remark made immediately prior to Section 3.6.1 and the results of the numerical investigation in Section 3.5.1, we feel that a more formal investigation into the performance of the Gittins index policy in such a machine environment may be of interest.

Finally, consider the discounted branching bandit model of Chapter 4, where the number of available machines is a stochastic process. We note that in the model presented in Section 4.2.1 (internal and external) arrivals into the system only come about when a machine processes a job or is up and idling. We suggest an extension of this model to allow arrivals to occur when machines are down.

References

Bellman, R. E. (1957). Dynamic Programming. Princeton: Princeton University Press.

Bertsimas, D. and Niño-Mora, J. (1996). Conservation laws, extended polymatroids and multi-armed bandit problems: a polyhedral approach to indexable systems. *Math. Oper. Res.*, **21**, 257-306.

Birge, J. B. G., Mittenthal, J. and Rinnooy Kan, A.H.G. (1990). Single machine scheduling subject to stochastic breakdowns. *Nav. Res. Logist.*, **37**, 660-677.

Blackwell, D. (1962). Discrete dynamic programming. *Ann. Math. Stat.*, **33**, 719-726.

Coffman, E, and Mitrani, I. (1980). A characterization of waiting time performance realisations by single server queues. *Oper. Res.*, **28**, 810-821.

Dacre, M., Glazebrook, K. D. and Niño-Mora, J. (1999). The achievable region approach to the optimal control of stochastic systems (with discussion). *J. R. Statist. Soc.*, **B61**, 747-791.

Denardo, E. V. and Miller, B. L. (1968). An optimality criterion for discrete dynamic programming with no discounting. *Ann. Math. Stat.*, **39**, 1220-1227.

Edmonds, J. (1970). Submodular functions, matroids and certain polyhedra. *Proceedings of Calgary International Conference on Combinatorial Structures and Their Applications*, R. Guy, H. Hanani, N. Sauer and J. Schönheim, (eds.), Gordon and Breach, New York, 69-87.

Federgruen, A. and Groenevelt, H. (1988a). Characterization and optimisation of achievable performance in general queueing systems. *Oper. Res.*, **36**, 733-741.

Federgruen, A. and Groenevelt, H. (1988b). M/G/c queueing systems with multiple customer classes: characterization and control of achievable performance under non-preemptive priority rules. *Mgmt. Sci.*, **34**, 1121-1138.

Gelenbe, E. and Mitrani, I. (1980). Analysis and Synthesis of Computer Systems: London: Academic Press.

Gittins, J. C. (1979). Bandit processes and dynamic allocation indices (with discussion). *J. R. Statist. Soc.*, **B41**, 148-177.

Gittins, J. C. and Jones, D. M. (1974). A dynamic allocation index for the sequential design of experiments. *Progress in Statistics: European Meeting of Statisticians, Budapest, 1972*, J. Gani, K. Sarkadi and I. Vince, (eds.), North-Holland, Amsterdam, 241-266.

Glazebrook, K. D. (1976). Stochastic scheduling with order constraints. *Int. J. Systems Sci.*, **7**, 657-666.

Glazebrook, K. D. (1982). On the evaluation of suboptimal strategies for families of alternative bandit processes. *J. Appl. Prob.*, **16**, 716-722.

Glazebrook, K. D. (1984). Scheduling stochastic jobs on a single machine subject to breakdowns. *Nav. Res. Logist. Quart.*, **31**, 251-264.

Glazebrook, K. D. (1987). Evaluating the effects of machine breakdowns in stochastic scheduling problems. *Nav. Res. Logist.*, **34**, 319-335.

Glazebrook, K. D. and Garbe, R. (1999). Almost optimal policies for stochastic systems which almost satisfy conservation laws. *Ann. Operat. Res.*, **92**, 19-43.

Glazebrook, K. D. and Niño-Mora, J. (1997). Scheduling multi-class queueing networks on parallel servers: approximate and heavy-traffic optimality of Klimov's rule. R. Burkard and G. Woeginger, (eds.), *Algorithms-ESA 97*, volume 1284 of *Springer Notes in Computer Science*, 232-245

Glazebrook, K. D. and Niño-Mora, J. (2001). Parallel scheduling of multiclass $M/M/m$ queues: approximate and heavy-traffic optimization of achievable performance. *Oper. Res.*, **49**, 609-623.

Glazebrook, K. D. and Wilkinson, D. J. (2000). Index-based policies for discounted multi-armed bandits on parallel machines. *Ann. Appl. Prob.*, **10**, 877-896.

Katehakis, M. N. and Veinott Jr., A. F. (1987). The multi-armed bandit problem: decomposition and computation. *Math. Oper. Res.*, **12**, 262-268.

Pinedo, M. and Rammouz, E. (1988). A note on stochastic scheduling on a single machine subject to breakdown and repair. *Prob. Eng Inf. Sci.*, **2**, 41-49.

Puterman, M. (1994). Markov Decision Processes: Discrete Stochastic Dynamic Programming, Wiley, New York.

Ross, S. M. (1970). Applied Probability Model with Optimisation Applications. San Francisco: Holden-Day.

Shanthikumar, J. G. and Yao, D. D. (1992). Multi-class queueing systems: Polymatroidal structure and optimal scheduling control. *Oper. Res.*, **40**, S293-S299.

Tsoucas, P. (1991). The Region of Achievable Performance in a Model of Klimov. IBM T. J. Watson Research Center: Technical Report RC16543.

Veinott, Jr., A. F. (1966). On finding optimal policies in discrete dynamic programming with no discounting. *Ann. Math. Stat.*, **37**, 1284-1294.

Weber, R.R. (1982). Scheduling jobs with stochastic processing requirements on parallel machines to minimize makespan or flowtime. *J. Appl. Prob.*, **19**, 167-182.

Weber, R.R., Varaiya, P. and Walrand, J. (1986). Scheduling jobs with stochastically ordered processing times on parallel machines to minimize expected flowtime. *J. Appl. Prob.*, **23**, 841-847.

Weiss, G. (1982). Multiserver stochastic scheduling. *Deterministic and Stochastic Scheduling*, M. A. H. Dempster, J. K. Lenstra and A. H. G. Rinnooy Kan, (eds.), 157-179, D. Reidel, Dordrecht.

Weiss, G. (1988). Branching bandit processes. *Prob. Eng. Inf. Sci.*, **2**, 269-278.

Weiss, G. (1990). Approximation results in parallel machines stochastic scheduling. *Ann. Oper. Res., Special Volume on Production Planning and Scheduling*, M. Queyranne, (ed.), **26**, 195-242.

Weiss, G. (1992). Turnpike optimality of Smith's rule in parallel machines stochastic scheduling. *Math. Oper. Res.*, **17**, 255-270.

Weiss, G. (1995). A tutorial in stochastic scheduling. *Scheduling Theory and its Applications*, P. Chrétienne, E. G. Coffman Jr, J. K. Lenstra and Z. Liu, (eds.), 33-64, Wiley, New York.

Whittle, P. (1980). Multi-armed bandits and the Gittins index. *J. R. Statist. Soc.*, **B42**, 143-149.

Whittle, P. (1981). Arm acquiring bandits. *Ann. Prob.*, **9**, 284-292.

Appendix A

```
program spd_block_limit
implicit none
integer :: i,j,k,l,h,d,b,type,count,countcomp,E,SIZE
integer :: SIM,num,RANDOM,y,cond,SPD,c,bandit1
integer,dimension(3) :: seed
integer,dimension(4):: allocind,allocopt
integer,dimension(4) :: argmax,STATE
double precision :: SMALL,BIG,MEAN,MEDIAN,STD,t,alpha,speed1,speed2,Z,sum,sum2
double precision :: Ropt,Rind,PERCENT,NUMBER
double precision, dimension(4) :: r,G,IND
double precision, dimension(4,4) :: P,Rmat
integer, dimension(15,4) :: S
double precision, dimension(15,4) :: A,AV
double precision, dimension(4,4) :: INDEX,INDEX2
double precision, dimension(4,4,4) :: Pmat
double precision, dimension(50) :: Subopt,ORDER

type = 7
SIM = 50
E = 4
seed = (/13623,5754,21459/)

do j = 15,1,-1
  d = j
  do i = 3,0,-1
    t = d/2**i
    if (t < 1) S(j,(4-i)) = 0
    if (t>= 1) then
      S(j,(4-i)) = 1
      d = mod(d,2**i)
    end if
  end do
end do

do SPD = 1,6
  if (SPD == 1) then
    speed1 = 6.0d0
    speed2 = 1.0d0
  else if (SPD == 2) then
    speed1 = 5.0d0
    speed2 = 2.0d0
  else if (SPD == 3) then
    speed1 = 4.5d0
    speed2 = 3.5d0
  else if (SPD == 4) then
    speed1 = 4.0d0
```

```
  speed2 = 3.0d0
else if (SPD == 5) then
  speed1 = 3.75d0
  speed2 = 3.25d0
else if (SPD == 6) then
  speed1 = 3.625d0
  speed2 = 3.375d0
end if

  do c = 1,7
    if (c == 1) then
      alpha = 0.5d0
    else if (c == 2) then
      alpha = 0.25d0
    else if (c == 3) then
      alpha = 0.1d0
    else if (c == 4) then
      alpha = 0.05d0
    else if (c == 5) then
      alpha = 0.025d0
    else if (c == 6) then
      alpha = 0.01d0
    else if (c == 7) then
      alpha = 0.005d0
    end if

    Subopt = 0d0
    Order = 0d0
    SMALL = 1.0e+20
    BIG = -1*(1.0e+20)
    NUMBER = 0d0

    do y = 1,SIM
      Ropt = 0d0
      Rind = 0d0

      call GETrandom(seed,RANDOM)

      h = 0
      do i = 1,4
        do j = 1,4
          do k = 1,4
            do l = 1,4
              h = h + 1
              if (h == RANDOM) then
                STATE = (/i,j,k,l/)
              end if
            end do
          end do
        end do
      end do

      call GETPmatrix(seed,Pmat)
      call GETRmatrix(seed,Rmat)

      do b = 1,4
        do i = 1,4
          do j = 1,4
            P(i,j) = Pmat(b,i,j)
         end do
```

```
      r(i) = Rmat(b,i)
    end do
    h = 15
    G = 0
    IND = 0
    argmax = 0
    A = 0
    A(15,:) = (/1,1,1,1/)
    AV = 0
    do k = 1,E
      if (k > 1) call GETsubset(E,argmax,k,h)
        count = 0
        do i = 1,E
          count = count + S(h,i)
          countcomp = E - count
        end do
        if (k > 1) call GETAmatrix_LIMIT(AV,S,count,countcomp,P,h,A)
        call GETindices(b,k,S,h,r,A,argmax,G,IND,INDEX)
    end do
  end do

  INDEX2 = INDEX
  allocopt = 0
  allocind = 0

call Getbandit1(INDEX2,bandit1,cond)
call GETrewards
(STATE,Rmat,Pmat,INDEX2,alpha,speed1,speed2,bandit1,Ropt,Rind)

  Subopt(y) = Ropt - Rind

  if(Subopt(y) >= BIG) then
    BIG = Subopt(y)
  end if
  if(Subopt(y) <= SMALL) then
    SMALL = Subopt(y)
  end if

end do

ORDER = 1.0e+20

do j = 1,SIM
  do i = 1,SIM
    if(ORDER(j) >= Subopt(i)) then
      ORDER(j) = Subopt(i)
      k = i
    end if
  end do
  Subopt(k) = 1.0e+21
end do

j = 0
do i = 1,SIM
  if(ORDER(i) > 1.0e-7) j = j + 1
end do
if (mod(j,2) == 1 .and. j > 0)then
  MEDIAN = ORDER(SIM - j + ((j+1)/2))
else if (mod(j,2) == 0 .and. j > 0)then
  MEDIAN = (ORDER(SIM - j + (j/2)) + ORDER(SIM - j + ((j/2)+1)))/2d0
```

```
      else if (j == 0) then
        MEDIAN = 0d0
      end if

      sum = 0d0
      sum2 = 0d0
      do i = 1,SIM
        sum = sum + ORDER(i)
        sum2 = sum2 + (ORDER(i))**2
      end do

      MEAN = sum/SIM*1d0
      t = (((1d0/SIM*1d0)*sum2)) - (MEAN**2)
      if(t > 0)then
        STD = sqrt((((1d0/SIM*1d0)*sum2)) - (MEAN**2))
      else
        STD = 0d0
      end if
      PERCENT = 100.0d0 - (100.0d0*j/SIM*1.0d0)
      NUMBER = 100.0d0 - (100.0d0*NUMBER/SIM*1.0d0)

 write(unit = 6, fmt= "(a)")"-----------------------------------------"
 write(unit = 6, fmt= "(a,f6.4,a,f6.4,a,f5.4)")
 "speed1 = ",speed1," speed2 = " ,speed2," alpha = " ,alpha
 write(unit = 6, fmt= "(a)")"-----------------------------------------"
 write(unit = 6, fmt= "(a,i3)")"     THE NUMBER OF SIMULATIONS IS  ", SIM
 write(unit = 6, fmt= "(a,f12.6,a,f12.6,a,f12.6)")
 "(1) Minimum", SMALL,"     (2) Median ",MEDIAN,"     (3) Maximum",BIG
 write(unit = 6, fmt= "(a,f12.6,a,f12.6,a,f12.6)")
 "(4) Mean   ", MEAN,"     (5) Std Dev",STD,"     (6) Percent",PERCENT
 write(unit = 6, fmt= "(a)")"========================================="

 open(unit = 7, file = "BLOCK_FINAL.dat")

 write(unit = 7, fmt="(a)")"-----------------------------------------"
 write(unit = 7, fmt= "(a,f6.4,a,f6.4,a,f5.4)")
 "speed1 = ",speed1," speed2 = " ,speed2," alpha = " ,alpha
 write(unit = 7, fmt= "(a)")"-----------------------------------------"
 write(unit = 7, fmt= "(a,i3)")"     THE NUMBER OF SIMULATIONS IS  ",SIM
 write(unit = 7, fmt= "(a,f12.6,a,f12.6,a,f12.6)")
 "(1) Minimum", SMALL,"     (2) Median",MEDIAN,"    (3) Maximum",BIG
 write(unit = 7, fmt= "(a,f12.6,a,f12.6,a,f12.6)")
 "(4) Mean   ", MEAN,"     (5) Std Dev",STD,"    (6) Percent",PERCENT
 write(unit = 7, fmt= "(a)")"========================================="

    end do
end do
end program
!=====================================================================
subroutine GETsubset(E,argmax,k,h)
implicit none integer :: k,h,E
integer, dimension(4) :: argmax

h = h - 2**(E - argmax(k - 1))
return
end
!=====================================================================
subroutine GETAmatrix_LIMIT(AV,S,count,countcomp,P,h,A)
implicit none integer :: d,g,i,j,h,f,k,count,countcomp,E
double precision :: matsum,Z,sum
```

```
integer, dimension(15,4) :: S
double precision, dimension(4,4) :: P
double precision, dimension(15,4) :: A,AV
double precision, dimension(countcomp,countcomp) :: Mmat,Pmat
double precision, dimension(countcomp,1) :: Pvec,Psum,Product
double precision, dimension(count) :: V1,V2

E = 4
Mmat = 0d0
d = 0
do i =1,E
  if(S(h,i) == 0) then
    if(d < countcomp) d = d + 1
    Mmat(d,d) = 1 - P(i,i)
    g = d
    do j = 1,E
      if (i < j) then
        if (S(h,j) == 0) then
          if (g < countcomp) g = g + 1
          Mmat(d,g) = (-1.0d0)*P(i,j)
          Mmat(g,d) = (-1.0d0)*P(j,i)
        end if
      end if
    end do
  end if
end do
!-------------------------------------------------
Psum = 0d0
Pvec = 0d0
Pmat = 0d0
d = 0
do i = 1,E
  g = 0
  if (S(h,i) == 0) then
    if (d < countcomp) d = d + 1
    do j = 1,E
      if (S(h,j) == 0) then
        if(g < countcomp) g = g + 1
        Pmat(d,g) = P(i,j)
      end if
    end do
  end if
end do
!-------------------------------------------------
do i = 1,countcomp
  do j = 1,countcomp
    Psum(i,1) = Psum(i,1) + Pmat(i,j)
  end do
end do
do i = 1,countcomp
  Pvec(i,1) = (1 - Psum(i,1))
end do
!-------------------------------------------------
Pvec = 1 + Pvec
!-------------------------------------------------
call GETinverse(countcomp,Mmat)
Product = 0d0
do i = 1,countcomp
  sum = 0d0
  do j = 1,countcomp
```

```
    sum = sum + (Mmat(i,j)*Pvec(j,1))
  end do
  Product(i,1) = sum
end do
Pvec = Product
!-----------------------------------------------
f = 0
do k = 1,4
  if (S(h,k) == 0) then
    if (f < countcomp) f = f + 1
    AV(h,k) = Pvec(f,1)
  end if
end do
d = 0
do i = 1,E
  matsum = 0
  if (S(h,i) == 1) then
    if (d < count) d = d + 1
    do j = 1,E
      if (S(h,j) == 0) matsum = matsum + (P(i,j) * AV(h,j))
    end do
    V1(d) = matsum
    matsum = 0
    do j = 1,E
      if (S(h,j) == 1) matsum = matsum + (P(i,j))
    end do
    V2(d) = matsum
    AV(h,i) = V1(d) + V2(d)
  end if
end do AV(15,:) = (/1,1,1,1/)
f = 0
do k = 1,4
  if (S(h,k) == 1) then
    if (f < count) f = f + 1
    A(h,k) = AV(h,k)
  end if
end do
return
end
!=====================================================================
subroutine GETindices(b,k,S,h,r,A,argmax,G,IND,INDEX)
implicit none integer :: b,i,h,k,l,j,E
double precision :: max,sum
integer, dimension(4) :: argmax
double precision, dimension(4) :: r,G,IND
double precision, dimension(15,4) :: A
integer, dimension(15,4) :: S
double precision, dimension(4,4) :: INDEX

if (k == 1) then
  max = -1000000
  do i = 1,4
    if (r(i)/A(h,i) > max) then
      max = r(i)/A(h,i)
      argmax(1) = i
    end if
  end do
  G(argmax(1)) = max
  IND(argmax(1)) = G(argmax(1))
else
```

```
  E = 4
  max = -1000000
  do i = 1,4
    l = 15
    sum = 0
    if (S(h,i) == 1) then
      do j = 1,k-1
        sum = sum + A(l,i)*G(argmax(j))
        l = l - 2**(E - argmax(j))
      end do
      if ((r(i) - sum)/A(h,i) > max) then
        max = (r(i) - sum)/A(h,i)
        argmax(k) = i
      end if
    end if
  end do
  G(argmax(k)) = max
  IND(argmax(k)) = G(argmax(k)) + IND(argmax(k - 1))
end if
INDEX(b,argmax(k)) = IND(argmax(k))
return
end
!=======================================================================
subroutine GETinverse(countcomp,Mmat)
implicit none
integer :: i,j,countcomp
double precision :: a,b,c,d,determinant
double precision, dimension(countcomp,countcomp) :: Mmat
double precision, dimension(3,3) :: Cofactor

if (countcomp == 1) then
  Mmat(1,1) = 1d0/Mmat(1,1)
else if (countcomp == 2) then
  a = Mmat(1,1)
  b = Mmat(1,2)
  c = Mmat(2,1)
  d = Mmat(2,2)
  determinant = (a*d) - (b*c)
  Mmat(1,1) = d/determinant
  Mmat(1,2) = -1d0*(b/determinant)
  Mmat(2,1) = -1d0*(c/determinant)
  Mmat(2,2) = a/determinant
else if (countcomp == 3) then
Cofactor(1,1) = ((Mmat(2,2)*Mmat(3,3)) - (Mmat(2,3)*Mmat(3,2)))
Cofactor(1,2) = -1d0*((Mmat(2,1)*Mmat(3,3)) - (Mmat(2,3)*Mmat(3,1)))
Cofactor(1,3) = ((Mmat(2,1)*Mmat(3,2)) - (Mmat(2,2)*Mmat(3,1)))

Cofactor(2,1) = -1d0*((Mmat(1,2)*Mmat(3,3)) - (Mmat(1,3)*Mmat(3,2)))
Cofactor(2,2) = ((Mmat(1,1)*Mmat(3,3)) - (Mmat(1,3)*Mmat(3,1)))
Cofactor(2,3) = -1d0*((Mmat(1,1)*Mmat(3,2)) - (Mmat(1,2)*Mmat(3,1)))

Cofactor(3,1) = ((Mmat(1,2)*Mmat(2,3)) - (Mmat(1,3)*Mmat(2,2)))
Cofactor(3,2) = -1d0*((Mmat(1,1)*Mmat(2,3)) - (Mmat(1,3)*Mmat(2,1)))
Cofactor(3,3) = ((Mmat(1,1)*Mmat(2,2)) - (Mmat(1,2)*Mmat(2,1)))

determinant = Mmat(1,1)*Cofactor(1,1) + Mmat(1,2)*Cofactor(1,2)
            + Mmat(1,3)*Cofactor(1,3)
  do i = 1,3
    do j = 1,3
      Mmat(i,j) = Cofactor(j,i)/determinant
```

```
    end do
  end do
end if
return
end
!=====================================================================
subroutine GETPmatrix(seed,Pmat)
implicit none
integer :: i,j,k
integer, dimension(3) :: seed
double precision ::UNIFORM,sum,L,U
double precision, dimension(4,4,4) :: PMat

L = 0.1 U = 0.9
  do k = 1,4
    do i = 1,4
      do j = 1,4
        call GETuniform(seed,L,U,UNIFORM)
        PMat(k,i,j) = UNIFORM
      end do
    end do
  end do

  do k = 1,4
    do i = 1,4
      sum = 0d0
      do j = 1,4
        sum = sum + PMat(k,i,j)
      end do
      do j = 1,4
        PMat(k,i,j) = Pmat(k,i,j)*(1.0d0/sum)
      end do
    end do
  end do
return
end
!=====================================================================
subroutine GETRmatrix(seed,Rmat)
implicit none
integer :: i,j
integer, dimension(3) :: seed
double precision :: L,U,x
double precision, dimension(4,4) :: RMat

L = 2d0
U = 5d0
do i = 1,4
  do j = 1,4
    call GETuniform(seed,L,U,x)
    RMat(i,j) = x
  end do
end do
return
end
!=====================================================================
subroutine GETuniform(seed,L,U,x)
implicit none
integer, dimension(3) :: seed
double precision :: r,s,x,L,U
```

```
seed(1) = mod(171*seed(1),30269)
seed(2) = mod(172*seed(2),30307)
seed(3) = mod(170*seed(3),30323)

s = seed(1)*1d0/30269 + seed(2)*1d0/30307 + seed(3)*1d0/30323 r =
s - int(s)
x = L + (U-L)*r
return
end
!=====================================================================
subroutine GETrandom(seed,x)
implicit none
integer :: x
integer, dimension(3) :: seed
double precision :: r,s

seed(1) = mod(171*seed(1),30269)
seed(2) = mod(172*seed(2),30307)
seed(3) = mod(170*seed(3),30323)

s = seed(1)*1d0/30269 + seed(2)*1d0/30307 + seed(3)*1d0/30323 r =
s - int(s)
x = int(256*r)
return
end
!=====================================================================
subroutine GETbandit1(IND,bandit1,cond)
implicit none
integer :: i,j,c,cond,bandit1
double precision :: max
double precision, dimension(3) :: Gsub
double precision, dimension(4) :: min
double precision, dimension(4,4) :: IND

min = 0d0
do i= 1,4
  min(i) = 1000000d0
  do j = 1,4
    if (IND(i,j) < min(i)) then
      min(i) = IND(i,j)
    end if
  end do
end do

do i = 1,3
  max = -10000
  do j = 1,4
    if (min(j) > max) then
      max = min(j)
      Gsub(i) = min(j)
      c = j
    end if
  end do
  if (i == 1) then
    bandit1 = c
  end if
  min(c) = -100000
end do

if (Gsub(1) /= Gsub(2) .and. Gsub(2) /= Gsub(3) .and. Gsub(1) /= Gsub(3)) then
```

```
  cond = 1
else
  cond = 2
end if
return
end
!=====================================================================
subroutine GETrewards
(c,Rmat,Pmat,INDEX,alpha,speed1,speed2,bandit1,Rewblock,Rewpolicy)
implicit none integer :: a,b,e,f,h,i,j,k,l,m,n,q,bandit1,bandit2,u
integer,dimension(3) :: v,uv,vu integer,dimension(4) :: c,IS,cnew
double precision :: alpha,maxi,Rewblock,Rewpolicy,maximum,speed1,speed2
double precision, dimension(2) :: dis,max,d
double precision, dimension(6) :: rew
double precision, dimension(10) :: Psum
double precision, dimension(14) :: block
double precision, dimension(4,2) :: IP1
double precision, dimension(4,4) :: Rmat
double precision, dimension(4,4) :: INDEX
double precision, dimension(4,4,4) :: Pmat
double precision, dimension(4,4,2,2) :: OB1
double precision, dimension(4,4,4,2) :: IP2,POL2
double precision, dimension(4,4,4,4,2) :: IP
double precision, dimension(4,4,4,4,2,2) :: OB2,OB3
double precision, dimension(4,4,4,4,4,2,2) :: OBone
double precision, dimension(4,4,4,4,4,4,2) :: OBtwo

dis(1) = exp(-1*(alpha/speed1)) dis(2) = exp(-1*(alpha/speed2))

d(1) = (1.0d0/alpha)*(1.0d0 - exp(-1*(alpha/(1.0d0*speed1))))
d(2) = (1.0d0/alpha)*(1.0d0 - exp(-1*(alpha/(1.0d0*speed2))))

OB1 = 0d0
OB2 = 0d0
OB3 = 0d0
IP1 = 0d0
IP2 = 0d0
IP = 0d0
POL2 = 0d0
OBone = 0d0
OBtwo = 0d0

do q = 1,int((speed1*1d0)*(-1.0d0/alpha)*log((alpha*10e-8)/(5.0d0/d(1))))+1
  if (mod(q,2) == 1) then
    e = 1
    f = 2
  else if (mod(q,2) == 0) then
    e = 2
    f = 1
  end if
  do i = 1,4
    do j = 1,4
      do k = 1,4
        do l = 1,4
          IS = (/i,j,k,l/)
          max = -10000d0
          rew = 0d0
          do m = 1,4
            h = 0
            do b = 1,3
```

```
              PSum(b) = 0d0
            end do
!------both opt block and policy------
            do a = 1,4
              if (a /= m) then
                h = h + 1
                uv(h) = IS(a)
                vu(h) = a
              end if
!------optimal block 3:1 allocations------
              PSum(1) = PSum(1) + Pmat(m,IS(m),a)*OB1(m,a,1,e)
              PSum(2) = PSum(2) + Pmat(m,IS(m),a)*OB1(m,a,2,e)
!------index policy------
              if (m == bandit1) then
                PSum(3) = PSum(3) + Pmat(m,IS(m),a)*IP1(a,e)
              end if
            end do
!------optimal block 3:1 allocations------
            OB1(m,IS(m),1,f) = Rmat(m,IS(m)) + dis(1)*PSum(1)
            OB1(m,IS(m),2,f) = Rmat(m,IS(m)) + dis(2)*PSum(2)
!------index policy------
            if (m == bandit1) then
              IP1(IS(m),f) = Rmat(m,IS(m)) + dis(1)*PSum(3)
              maxi = -10000d0
              do b = 1,3
                if (INDEX(vu(b),uv(b)) > maxi ) then
                  maxi = INDEX(vu(b),uv(b))
                  bandit2 = vu(b)
                end if
              end do
            end if
            do a = 1,2
              max(a) = -10000d0
            end do
            h = 0
!------------------
            do n = 1,4
              if (m /= n) then
                h = h + 1
                do b = 4,10
                  PSum(b) = 0d0
                end do
!------------------
                do a = 1,4
                  v = uv
                  v(h) =a
!------optimal block 3:1 allocations------
                  PSum(4) = PSum(4) + Pmat(n,IS(n),a)*OB3(m,v(1),v(2),v(3),1,e)
                  PSum(5) = PSum(5) + Pmat(n,IS(n),a)*OB3(m,v(1),v(2),v(3),2,e)
!------optimal block 2:2 allocations------
                  if (m < n) then
                    PSum(6) = PSum(6) + Pmat(m,IS(m),a)*OB2(m,n,a,IS(n),1,e)
                    PSum(7) = PSum(7) + Pmat(n,IS(n),a)*OB2(m,n,IS(m),a,1,e)
                    PSum(8) = PSum(8) + Pmat(m,IS(m),a)*OB2(m,n,a,IS(n),2,e)
                    PSum(9) = PSum(9) + Pmat(n,IS(n),a)*OB2(m,n,IS(m),a,2,e)
                  end if
!------index policy------
                  if (m == bandit1 .and. n == bandit2) then
                    PSum(10) = PSum(10) + Pmat(n,IS(n),a)*IP2(v(1),v(2),v(3),e)
                  end if
```

```
                end do
!------optimal block 3:1 allocations------
                rew(1) = Rmat(n,IS(n)) + dis(2)*PSum(4)
                rew(2) = Rmat(n,IS(n)) + dis(1)*PSum(5)
!------optimal block 2:2 allocations------
                if (m < n) then
                  rew(3) = Rmat(m,IS(m)) + dis(1)*PSum(6)
                  rew(4) = Rmat(n,IS(n)) + dis(1)*PSum(7)
                  rew(5) = Rmat(m,IS(m)) + dis(2)*PSum(8)
                  rew(6) = Rmat(n,IS(n)) + dis(2)*PSum(9)
                  if (rew(3) > rew(4)) then
                    OB2(m,n,IS(m),IS(n),1,f) = rew(3)
                  else if (rew(3) <= rew(4)) then
                    OB2(m,n,IS(m),IS(n),1,f) = rew(4)
                  end if
                  if (rew(5) > rew(6)) then
                    OB2(m,n,IS(m),IS(n),2,f) = rew(5)
                  else if (rew(5) <= rew(6)) then
                    OB2(m,n,IS(m),IS(n),2,f) = rew(6)
                  end if
                end if
!------index policy------
                if (m == bandit1 .and. n == bandit2) then
                  IP2(uv(1),uv(2),uv(3),f) = Rmat(n,IS(n)) + dis(2)*PSum(10)
                  IP(i,j,k,l,f) = IP1(IS(m),f) + IP2(uv(1),uv(2),uv(3),f)
                end if
                do a = 1,2
                  if (rew(a) > max(a)) then
                    max(a) = rew(a)
                  end if
                end do
              end if
            end do
!-----------------n do-----
!------optimal block 3:1 allocations------
            OB3(m,uv(1),uv(2),uv(3),1,f) = max(1)
            OB3(m,uv(1),uv(2),uv(3),2,f) = max(2)
!-----------------
OBone(m,i,j,k,l,1,f) = OB1(m,IS(m),1,f) + OB3(m,uv(1),uv(2),uv(3),1,f)
OBone(m,i,j,k,l,2,f) = OB1(m,IS(m),2,f) + OB3(m,uv(1),uv(2),uv(3),2,f)
          end do
!-----------------m do-----
OBtwo(1,2,i,j,k,l,f) = OB2(1,2,IS(1),IS(2),1,f) + OB2(3,4,IS(3),IS(4),2,f)
OBtwo(1,3,i,j,k,l,f) = OB2(1,3,IS(1),IS(3),1,f) + OB2(2,4,IS(2),IS(4),2,f)
OBtwo(1,4,i,j,k,l,f) = OB2(1,4,IS(1),IS(4),1,f) + OB2(2,3,IS(2),IS(3),2,f)
OBtwo(2,3,i,j,k,l,f) = OB2(2,3,IS(2),IS(3),1,f) + OB2(1,4,IS(1),IS(4),2,f)
OBtwo(2,4,i,j,k,l,f) = OB2(2,4,IS(2),IS(4),1,f) + OB2(1,3,IS(1),IS(3),2,f)
OBtwo(3,4,i,j,k,l,f) = OB2(3,4,IS(3),IS(4),1,f) + OB2(1,2,IS(1),IS(2),2,f)
        end do
      end do
    end do
  end do
if(q == int((speed2*1d0)*(-1.0d0/alpha)*log((alpha*10e-8)/(5.0d0/d(2))))+1) then
  POL2 = IP2
end if
end do
h = 0
do a = 1,4
  if (a /= bandit1) then
    h = h + 1
```

```
    cnew(h) = c(a)
    v(h) = a
  end if
end do

Rewpolicy = IP1(c(bandit1),f) + POL2(cnew(1),cnew(2),cnew(3),f)

block(1) = OBone(1,c(1),c(2),c(3),c(4),1,f)
block(2) = OBone(2,c(1),c(2),c(3),c(4),1,f)
block(3) = OBone(3,c(1),c(2),c(3),c(4),1,f)
block(4) = OBone(4,c(1),c(2),c(3),c(4),1,f)
block(5) = OBone(1,c(1),c(2),c(3),c(4),2,f)
block(6) = OBone(2,c(1),c(2),c(3),c(4),2,f)
block(7) = OBone(3,c(1),c(2),c(3),c(4),2,f)
block(8) = OBone(4,c(1),c(2),c(3),c(4),2,f)
block(9) = OBtwo(1,2,c(1),c(2),c(3),c(4),f)
block(10) = OBtwo(1,3,c(1),c(2),c(3),c(4),f)
block(11) = OBtwo(1,4,c(1),c(2),c(3),c(4),f)
block(12) = OBtwo(2,3,c(1),c(2),c(3),c(4),f)
block(13) = OBtwo(2,4,c(1),c(2),c(3),c(4),f)
block(14) = OBtwo(3,4,c(1),c(2),c(3),c(4),f)

maximum = -1000d0
do b = 1,14
  if(block(b) > maximum) then
    maximum = block(b)
  end if
end do

Rewblock = maximum
return
end
!=======================================================================
```

Appendix B

```
program spd_opt_limit_6&1

implicit none
integer :: i,j,k,l,h,d,b,type,speed1,speed2,count,countcomp,E,M,SIZE
integer :: num,RANDOM,y,cond,c,bandit1,SIM
integer, dimension(3) :: seed
integer, dimension(4) :: argmax,STATE
double precision :: SMALL,BIG,MEAN,MEDIAN,STD,t,alpha,Z,sum,sum2,PERCENT
double precision :: Ropt,Rind,Reward1,Reward2,Reward3
double precision, dimension(4) :: r,G,IND
double precision, dimension(4,4) :: P,Rmat
integer, dimension(15,4) :: S
double precision, dimension(15,4) :: A,AV
double precision, dimension(4,4) :: INDEX,INDEX2
double precision, dimension(4,4,4) :: Pmat
double precision, dimension(500) :: Subopt,ORDER

SIM = 500
type = 7
speed1 = 6
speed2 = 1
seed =(/21561,1821,1929/)
E = 4

do j = 15,1,-1
  d = j
  do i = 3,0,-1
    t = d/2**i
    if (t < 1) S(j,(4-i)) = 0
    if (t>= 1) then
      S(j,(4-i)) = 1
      d = mod(d,2**i)
    end if
  end do
end do

do c = 1,7
  if (c == 1) then
    alpha = 0.5d0
  else if (c == 2) then
    alpha = 0.25d0
  else if (c == 3) then
    alpha = 0.1d0
  else if (c == 4) then
    alpha = 0.05d0
  else if (c == 5) then
    alpha = 0.025d0
  else if (c == 6) then
```

```
      alpha = 0.01d0
    else if (c == 7) then
      alpha = 0.005d0
    end if

    Subopt = 0d0
    order = 0d0
    SMALL = 1.0e+20
    BIG = -1*(1.0e+20)

    do y = 1,SIM

      Ropt = 0d0
      Rind = 0d0
      Reward1 = 0d0
      Reward2 = 0d0

      call GETrandom(seed,RANDOM)

      h = 0
      do i = 1,4
        do j = 1,4
          do k = 1,4
            do l = 1,4
              h = h + 1
              if (h == RANDOM) then
                STATE = (/i,j,k,l/)
              end if
            end do
          end do
        end do
      end do

      call GETPmatrix(seed,Pmat)
      call GETRmatrix(seed,Rmat)
!     call GETPmatrix_structured(Pmat)
!     call GETRmatrix_structured(seed,Rmat)
      do b = 1,4
        do i = 1,4
          do j = 1,4
            P(i,j) = Pmat(b,i,j)
          end do
          r(i) = Rmat(b,i)
        end do
        h = 15
        G = 0
        IND = 0
        argmax = 0
        A = 0
        A(15,:) = (/1,1,1,1/)
        AV = 0
        do k = 1,E
          if (k > 1) call GETsubset(E,argmax,k,h)
            count = 0
            do i = 1,E
              count = count + S(h,i)
              countcomp = E - count
            end do
            if (k > 1) call GETAmatrix_LIMIT(AV,S,count,countcomp,P,h,A)
            call GETindices(b,k,S,h,r,A,argmax,G,IND,INDEX)
```

```
    end do
  end do

  INDEX2 = INDEX
  call Getbandit1(INDEX2,bandit1,cond)
  call GETrewards
  (STATE,Rmat,Pmat,INDEX2,alpha,speed1,speed2,bandit1,Ropt,Rind)

  Subopt(y) = Ropt - Rind

  if(Subopt(y) >= BIG) then
    BIG = Subopt(y)
  end if
  if(Subopt(y) <= SMALL) then
    SMALL = Subopt(y)
  end if

end do

ORDER = 1.0e+20

do j = 1,SIM
  do i = 1,SIM
    if(ORDER(j) >= Subopt(i)) then
      ORDER(j) = Subopt(i)
      k = i
    end if
  end do
  Subopt(k) = 1.0e+21
end do

j = 0
do i = 1,SIM
  if(ORDER(i) > 1.0e-7) j = j + 1
end do
if (mod(j,2) == 1 .and. j > 0)then
  MEDIAN = ORDER(SIM - j + ((j+1)/2))
else if (mod(j,2) == 0 .and. j > 0)then
  MEDIAN = (ORDER(SIM - j + (j/2)) + ORDER(SIM - j + ((j/2)+1)))/2d0
else if (j == 0) then
  MEDIAN = 0d0
end if

sum = 0d0
sum2 = 0d0
do i = 1,SIM
  sum = sum + ORDER(i)
  sum2 = sum2 + (ORDER(i))**2
end do

MEAN = sum/SIM*1d0
t = (((1d0/SIM*1d0)*sum2)) - (MEAN**2)
if(t > 0)then
  STD = sqrt((((1d0/SIM*1d0)*sum2)) - (MEAN**2))
else
  STD = 0d0
end if
PERCENT = 100.0d0 -(100d0*j/SIM*1d0)

write(unit = 6, fmt= "(a,i2,a,i2,a,i3)")
```

```
  "speed1 = ",speed1," speed2 = " ,speed2," THE NUMBER OF SIMULATIONS IS  ",SIM
  write(unit = 6, fmt= "(a)")" "
  write(unit = 6, fmt= "(a,f6.4)")"alpha = " ,alpha
  write(unit = 6, fmt= "(a)")"========================================="
  write(unit = 6, fmt= "(a,f12.6,a,f12.6,a,f12.6)")
  "(1) Minimum", SMALL,"     (2) Median ",MEDIAN,"    (3) Maximum",BIG
  write(unit = 6, fmt= "(a,f12.6,a,f12.6,a,f12.6)")
  "(4) Mean   ", MEAN,"     (5) Std Dev",STD,"    (6) Percent",PERCENT
  write(unit = 6, fmt= "(a)")"========================================="

  open(unit = 7, file = "OPT_FINAL_6&1.dat")
!  open(unit = 7, file = "OPT_STRUCTURED_6&1.dat")

  write(unit = 7, fmt= "(a,i2,a,i2,a,i3)")
  "speed1 = ",speed1," speed2 = " ,speed2," THE NUMBER OF SIMULATIONS IS  ",SIM
  write(unit = 7, fmt= "(a)")" "
  write(unit = 7, fmt= "(a,f6.4)")"alpha = " ,alpha
  write(unit = 7, fmt= "(a)")"========================================="
  write(unit = 7, fmt= "(a,f12.6,a,f12.6,a,f12.6)")
  "(1) Minimum", SMALL,"     (2) Median ",MEDIAN,"    (3) Maximum",BIG
  write(unit = 7, fmt= "(a,f12.6,a,f12.6,a,f12.6)")
  "(4) Mean   ", MEAN,"     (5) Std Dev",STD,"    (6) Percent",PERCENT
  write(unit = 7, fmt= "(a)")"========================================="
end do

end program
!=====================================================================
subroutine GETsubset(E,argmax,k,h)
implicit none
integer :: k,h,E
integer, dimension(4) :: argmax

h = h - 2**(E - argmax(k - 1))
return
end
!=====================================================================
subroutine GETAmatrix_LIMIT(AV,S,count,countcomp,P,h,A)
implicit none
integer :: d,g,i,j,h,f,k,count,countcomp,E
double precision :: matsum,Z,sum
integer, dimension(15,4) :: S
double precision, dimension(4,4) :: P
double precision, dimension(15,4) :: A,AV
double precision, dimension(countcomp,countcomp) :: Mmat,Pmat
double precision, dimension(countcomp,1) :: Pvec,Psum,Product
double precision, dimension(count) :: V1,V2

E = 4
!-----------------------------------------------
Mmat = 0d0
!-----------------------------------------------
d = 0
do i = 1,E
  if(S(h,i) == 0) then
    if(d < countcomp) d = d + 1
    Mmat(d,d) = 1 - P(i,i)
    g = d
    do j = 1,E
      if (i < j) then
        if (S(h,j) == 0) then
```

```
          if (g < countcomp) g = g + 1
          Mmat(d,g) = (-1.0d0)*P(i,j)
          Mmat(g,d) = (-1.0d0)*P(j,i)
        end if
      end if
    end do
  end if
end do
!-------------------------------------------------
Psum = 0d0
Pvec = 0d0
Pmat = 0d0
d = 0
do i = 1,E
  g = 0
  if (S(h,i) == 0) then
    if (d < countcomp) d = d + 1
    do j = 1,E
      if (S(h,j) == 0) then
        if(g < countcomp) g = g + 1
        Pmat(d,g) = P(i,j)
      end if
    end do
  end if
end do
!-------------------------------------------------
do i = 1,countcomp
  do j = 1,countcomp
    Psum(i,1) = Psum(i,1) + Pmat(i,j)
  end do
end do
do i = 1,countcomp
  Pvec(i,1) = (1 - Psum(i,1))
end do
!-------------------------------------------------
Pvec = 1 + Pvec
!-------------------------------------------------
call GETinverse(countcomp,Mmat)
Product = 0d0
do i = 1,countcomp
  sum = 0d0
  do j = 1,countcomp
    sum = sum + (Mmat(i,j)*Pvec(j,1))
  end do
  Product(i,1) = sum
end do Pvec = Product
!-------------------------------------------------
f = 0
do k = 1,4
  if (S(h,k) == 0) then
    if (f < countcomp) f = f + 1
    AV(h,k) = Pvec(f,1)
  end if
end do

d = 0
do i = 1,E
  matsum = 0
  if (S(h,i) == 1) then
    if (d < count) d = d + 1
```

```
    do j = 1,E
      if (S(h,j) == 0) matsum = matsum + (P(i,j) * AV(h,j))
    end do
    V1(d) = matsum
    matsum = 0
    do j = 1,E
      if (S(h,j) == 1) matsum = matsum + (P(i,j))
    end do
    V2(d) = matsum
    AV(h,i) = V1(d) + V2(d)
  end if
end do

AV(15,:) = (/1,1,1,1/)

f = 0
do k = 1,4
  if (S(h,k) == 1) then
    if (f < count) f = f + 1
    A(h,k) = AV(h,k)
  end if
end do
return
end
!=====================================================================
subroutine GETindices(b,k,S,h,r,A,argmax,G,IND,INDEX)
implicit none
integer :: b,i,h,k,l,j,E
double precision :: max,sum
integer, dimension(4) :: argmax
double precision, dimension(4) :: r,G,IND
double precision, dimension(15,4) :: A
integer, dimension(15,4) :: S
double precision, dimension(4,4) :: INDEX

if (k == 1) then
  max = -1000000
  do i = 1,4
    if (r(i)/A(h,i) > max) then
      max = r(i)/A(h,i)
      argmax(1) = i
    end if
  end do
  G(argmax(1)) = max
  IND(argmax(1)) = G(argmax(1))
else
  E = 4
  max = -1000000
  do i = 1,4
    l = 15
    sum = 0
    if (S(h,i) == 1) then
      do j = 1,k-1
        sum = sum + A(l,i)*G(argmax(j))
        l = l - 2**(E - argmax(j))
      end do
      if ((r(i) - sum)/A(h,i) > max) then
        max = (r(i) - sum)/A(h,i)
        argmax(k) = i
      end if
```

```
    end if
  end do
  G(argmax(k)) = max
  IND(argmax(k)) = G(argmax(k)) + IND(argmax(k - 1))
end if

INDEX(b,argmax(k)) = IND(argmax(k))
return
end
!=============================================================
subroutine GETinverse(countcomp,Mmat)
implicit none
integer :: i,j,countcomp
double precision :: a,b,c,d,determinant
double precision, dimension(countcomp,countcomp) :: Mmat
double precision, dimension(3,3) :: Cofactor

if (countcomp == 1) then
  Mmat(1,1) = 1d0/Mmat(1,1)
else if (countcomp == 2) then
  a = Mmat(1,1)
  b = Mmat(1,2)
  c = Mmat(2,1)
  d = Mmat(2,2)
  determinant = (a*d) - (b*c)
  Mmat(1,1) = d/determinant
  Mmat(1,2) = -1d0*(b/determinant)
  Mmat(2,1) = -1d0*(c/determinant)
  Mmat(2,2) = a/determinant
else if (countcomp == 3) then
  Cofactor(1,1) = ((Mmat(2,2)*Mmat(3,3)) - (Mmat(2,3)*Mmat(3,2)))
  Cofactor(1,2) = -1d0*((Mmat(2,1)*Mmat(3,3)) - (Mmat(2,3)*Mmat(3,1)))
  Cofactor(1,3) = ((Mmat(2,1)*Mmat(3,2)) - (Mmat(2,2)*Mmat(3,1)))

  Cofactor(2,1) = -1d0*((Mmat(1,2)*Mmat(3,3)) - (Mmat(1,3)*Mmat(3,2)))
  Cofactor(2,2) = ((Mmat(1,1)*Mmat(3,3)) - (Mmat(1,3)*Mmat(3,1)))
  Cofactor(2,3) = -1d0*((Mmat(1,1)*Mmat(3,2)) - (Mmat(1,2)*Mmat(3,1)))

  Cofactor(3,1) = ((Mmat(1,2)*Mmat(2,3)) - (Mmat(1,3)*Mmat(2,2)))
  Cofactor(3,2) = -1d0*((Mmat(1,1)*Mmat(2,3)) - (Mmat(1,3)*Mmat(2,1)))
  Cofactor(3,3) = ((Mmat(1,1)*Mmat(2,2)) - (Mmat(1,2)*Mmat(2,1)))

  determinant = Mmat(1,1)*Cofactor(1,1) + Mmat(1,2)*Cofactor(1,2)
              + Mmat(1,3)*Cofactor(1,3)

  do i = 1,3
    do j = 1,3
      Mmat(i,j) = Cofactor(j,i)/determinant
    end do
  end do
end if
return
end
!=============================================================
subroutine GETPmatrix(seed,Pmat)
implicit none
integer :: i,j,k
integer, dimension(3) :: seed
double precision ::UNIFORM,sum,L,U
double precision, dimension(4,4,4) :: PMat
```

```
L = 0.1
U = 0.9

  do k = 1,4
    do i = 1,4
      do j = 1,4
        call GETuniform(seed,L,U,UNIFORM)
        PMat(k,i,j) = UNIFORM
      end do
    end do
  end do

  do k = 1,4
    do i = 1,4
      sum = 0d0
      do j = 1,4
        sum = sum + PMat(k,i,j)
      end do
      do j = 1,4
        PMat(k,i,j) = Pmat(k,i,j)*(1.0d0/sum)
      end do
    end do
  end do
return
end
!================================================================
subroutine GETPmatrix_structured(Pmat)
implicit none
integer :: i,j,k
double precision, dimension(4,4,4) :: PMat

do k = 1,4
  do i = 1,4
    do j = 1,4
      PMat(k,i,j) = 0.0d0
    end do
  end do
end do

do k = 1,4
  PMat(k,1,2) = 1.0d0
  PMat(k,2,3) = 1.0d0
  PMat(k,3,4) = 1.0d0
  PMat(k,4,1) = 1.0d0
end do
return
end
!================================================================
subroutine GETRmatrix(seed,Rmat)
implicit none
integer :: i,j
integer, dimension(3) :: seed
double precision :: L,U,x
double precision, dimension(4,4) :: RMat

L = 2d0
U = 5d0

do i = 1,4
```

```
  do j = 1,4
    call GETuniform(seed,L,U,x)
    RMat(i,j) = x
  end do
end do
return
end
!=============================================================
subroutine GETRmatrix_structured(seed,Rmat)
implicit none
integer :: i,j
integer, dimension(3) :: seed
double precision :: L,U,x
double precision, dimension(4,4) :: RMat

L = 0.0001d0
U = 0.0002d0

do i = 1,4
  do j = 2,4
    call GETuniform(seed,L,U,x)
    RMat(i,j) = x
  end do
end do

L = 2d0 U = 5d0

do i = 1,4
  call GETuniform(seed,L,U,x)
  RMat(i,1) = x
end do
return
end
!=============================================================
subroutine GETuniform(seed,L,U,x)
implicit none
integer, dimension(3) :: seed
double precision :: r,s,x,L,U

seed(1) = mod(171*seed(1),30269)
seed(2) = mod(172*seed(2),30307)
seed(3) = mod(170*seed(3),30323)

s = seed(1)*1d0/30269 + seed(2)*1d0/30307 + seed(3)*1d0/30323
r = s - int(s)

x = L + (U-L)*r
return
end
!=============================================================
subroutine GETrandom(seed,x)
implicit none
integer :: x
integer, dimension(3) :: seed
double precision :: r,s

seed(1) = mod(171*seed(1),30269)
seed(2) = mod(172*seed(2),30307)
seed(3) = mod(170*seed(3),30323)
```

```
s = seed(1)*1d0/30269 + seed(2)*1d0/30307 + seed(3)*1d0/30323
r = s - int(s)

x = int(256*r)
return
end
!=============================================================
subroutine Check(IND,num)
implicit none
integer :: i,j,k,c,d,num
double precision :: max
integer, dimension(4,4) :: ORDER1,ORDER2
double precision, dimension(4,4) :: IND

c = 0
d = 0
ORDER1 = 0
do k = 1,16
  max = -100000
  do i = 1,4
    do j = 1,4
      if (IND(i,j) > max) then
        max = IND(i,j)
        c = i
        d = j
      end if
    end do
  end do
  ORDER1(c,d) = k
  IND(c,d) = 0
end do

c = 0
d = 0
ORDER2 = 0
do k = 1,16
  max = -100000
  do i = 1,4
    do j = 1,4
      if (IND(i,j) > max) then
        max = IND(i,j)
        c = i
        d = j
      end if
    end do
  end do
  ORDER2(c,d) = k
  IND(c,d) = 0
end do

num = 0
do i = 1,4
  do j = 1,4
    if(ORDER1(i,j) /= ORDER2(i,j)) then
      num = num + 1
    end if
  end do
end do
return
end
```

```
!=============================================================
subroutine GETbandit1(IND,bandit1,cond)
implicit none
integer ::i,j,c,cond,bandit1
double precision :: max
double precision, dimension(3) :: Gsub
double precision, dimension(4) :: min
double precision, dimension(4,4) :: IND

min = 0d0
do i= 1,4
  min(i) = 1000000d0
  do j = 1,4
    if (IND(i,j) < min(i)) then
      min(i) = IND(i,j)
    end if
  end do
end do

do i = 1,3
  max = -10000
  do j = 1,4
    if (min(j) > max) then
      max = min(j)
      Gsub(i) = min(j)
      c = j
    end if
  end do
  if (i == 1) then
    bandit1 = c
  end if
  min(c) = -100000
end do

if (Gsub(1) /= Gsub(2) .and. Gsub(2) /= Gsub(3) .and. Gsub(1) /= Gsub(3)) then
  cond = 1
else
  cond = 2
end if
return
end
!=====================================================================
subroutine
GETrewards(c,Rmat,Pmat,INDEX,alpha,speed1,speed2,bandit1,Rewoptimal,Rewpolicy)
implicit none
integer :: a,b,e,f,h,i,j,k,l,m,n,q,speed1,speed2,bandit1,bandit2,bplace
integer,dimension(3) :: v,uv,vu,cnew
integer,dimension(4) :: c,IS,u
double precision :: alpha,Rnm,maxi,Rewoptimal,Rewpolicy
double precision, dimension(2) :: dis
double precision, dimension(4) :: d
double precision, dimension(6) :: max,rew
double precision, dimension(8) :: Psum
double precision, dimension(4,2) :: IP1,POL1
double precision, dimension(4,4) :: Rmat double
precision, dimension(4,4) :: INDEX
double precision, dimension(4,4,4) :: Pmat
double precision, dimension(4,4,4,2) ::
IP2,POL2 double precision, dimension(4,4,4,4,2) :: O0,IP
double precision, dimension(4,4,4,4,4,2) :: O1,O2,O3,O4,O5
```

```
d(1) = (1.0d0/alpha)*(1.0d0 - exp(-1.0d0*(alpha/(1.0d0*speed1))))
d(2) = (1.0d0/alpha)*(1.0d0 - exp(-1.0d0*(alpha/(1.0d0*speed2))))
d(3) = (1.0d0/alpha)*(1.0d0 - exp(-1.0d0*(alpha/(1.0d0*speed1*speed2))))
d(4) = exp(-1.0d0*(alpha/(1.0d0*speed1*speed2)))

dis(1) = exp(-1.0d0*(alpha/(1.0d0*speed1)))
dis(2) = exp(-1.0d0*(alpha/(1.0d0*speed2)))

O0=0d0
O1=0d0
O2=0d0
O3=0d0
O4=0d0
O5=0d0
IP1=0d0
IP2=0d0
IP=0d0
POL1 = 0d0
POL2 = 0d0

do q = 1,int((1.0d0*speed1*speed2)*(-1.0d0/alpha)
         *log((alpha*10e-8)/((5.0d0/d(1))+(5.0d0/d(2)))))+1
  if (mod(q,2) == 1) then
    e = 1
    f = 2
  else if (mod(q,2) == 0) then
    e = 2
    f = 1
  end if
  do i = 1,4
    do j = 1,4
      do k = 1,4
        do l = 1,4
          IS = (/i,j,k,l/)
          max(6) = -10000d0
          rew = 0d0
          do m = 1,4
            do a = 1,5
              max(a) = -10000d0
            end do
!--------------------------
            if (m == bandit1) then
              h = 0
              Psum(7) = 0d0
              do a = 1,4
                if (a /= bandit1) then
                  h = h + 1
                  uv(h) = IS(a)
                  vu(h) = a
                end if
                Psum(7) = Psum(7) + Pmat(m,IS(m),a)*IP1(a,e)
              end do
IP1(IS(bandit1),f) = Rmat(bandit1,IS(bandit1)) + dis(1)*Psum(7)
              maxi = -10000d0
              bandit2 = 0
              bplace = 0
              do b = 1,3
                if (INDEX(vu(b),uv(b)) > maxi) then
                  maxi = INDEX(vu(b),uv(b))
```

```
                bandit2 = vu(b)
                bplace = b
              end if
            end do
          end if
          h = 0
!---------------------------
          do n = 1,4
            if (n /= m) then
!---------------------------
              h = h + 1
!---------------------------
              Rnm = ((Rmat(n,IS(n))/d(1)) + (Rmat(m,IS(m))/d(2)))*d(3)
              do b = 1,8
                PSum(b) = 0d0
              end do
              do a = 1,4
!---------------------------
                if (m == bandit1 .and. n == bandit2) then
                  v = uv
                  v(bplace) = a
Psum(8) = Psum(8) + Pmat(bandit2,IS(bandit2),a)*IP2(v(1),v(2),v(3),e)
                end if
!---------------------------
                u = IS
                u(n) = a
Psum(1) = Psum(1) + Pmat(n,IS(n),a)*O1(u(1),u(2),u(3),u(4),m,e)
Psum(2) = Psum(2) + Pmat(n,IS(n),a)*O2(u(1),u(2),u(3),u(4),m,e)
Psum(3) = Psum(3) + Pmat(n,IS(n),a)*O3(u(1),u(2),u(3),u(4),m,e)
Psum(4) = Psum(4) + Pmat(n,IS(n),a)*O4(u(1),u(2),u(3),u(4),m,e)
Psum(5) = Psum(5) + Pmat(n,IS(n),a)*O5(u(1),u(2),u(3),u(4),m,e)
                do b = 1,4
                  u(m) = b
Psum(6) = Psum(6) + Pmat(n,IS(n),a)*Pmat(m,IS(m),b)*O0(u(1),u(2),u(3),u(4),e)
                end do
              end do
              rew(1) = Rnm + d(4)*PSum(2)
              rew(2) = Rnm + d(4)*PSum(3)
              rew(3) = Rnm + d(4)*PSum(4)
              rew(4) = Rnm + d(4)*PSum(5)
              rew(5) = Rnm + d(4)*PSum(6)
              rew(6) = Rnm + d(4)*PSum(1)
!-----------------------
              if (m == bandit1 .and. n == bandit2) then
IP2(uv(1),uv(2),uv(3),f) = Rmat(bandit2,IS(bandit2)) + dis(2)*Psum(8)
              end if
!-----------------------
              do a = 1,6
                if (rew(a) > max(a)) then
                  max(a) = rew(a)
                end if
              end do
            end if
          end do
          O1(i,j,k,l,m,f) = max(1)
          O2(i,j,k,l,m,f) = max(2)
          O3(i,j,k,l,m,f) = max(3)
          O4(i,j,k,l,m,f) = max(4)
          O5(i,j,k,l,m,f) = max(5)
        end do
```

```
          OO(i,j,k,l,f) = max(6)
        end do
      end do
    end do
  end do

  if(q == int((speed2*1d0)*(-1.0d0/alpha)
          *log((alpha*10e-8)/(5.0d0/d(2))))+1) then
  POL2 = IP2
  end if
  if(q == int((speed1*1d0)*(-1.0d0/alpha)
          *log((alpha*10e-8)/(5.0d0/d(1))))+1) then
  POL1 = IP1
  end if

end do

Rewoptimal = OO(c(1),c(2),c(3),c(4),f)
h = 0
do a = 1,4
  if (a /= bandit1) then
    h = h + 1
    cnew(h) = c(a)
    v(h) = a
  end if
end do

Rewpolicy = POL1(c(bandit1),f) + POL2(cnew(1),cnew(2),cnew(3),f)
return
end
!=============================================================================
```

Appendix C

```
program spd_opt_limit_5&2

implicit none
integer :: i,j,k,l,h,d,b,type,speed1,speed2,count,countcomp,E,M,SIZE
integer :: num,RANDOM,y,cond,c,bandit1,SIM
integer, dimension(3) :: seed
integer, dimension(4) :: argmax,STATE
double precision :: SMALL,BIG,MEAN,MEDIAN,STD,t,alpha,Z,sum,sum2,PERCENT
double precision :: Ropt,Rind,Reward1,Reward2,Reward3
double precision, dimension(4) :: r,G,IND
double precision, dimension(4,4) :: P,Rmat
integer, dimension(15,4) :: S
double precision, dimension(15,4) :: A,AV
double precision, dimension(4,4) :: INDEX,INDEX2
double precision, dimension(4,4,4) :: Pmat
double precision, dimension(500) :: Subopt,ORDER

type = 7
SIM = 500
speed1 = 5
speed2 = 2
seed =(/16849,2671,23494/)
E = 4

do j = 15,1,-1
  d = j
  do i = 3,0,-1
    t = d/2**i
    if (t < 1) S(j,(4-i)) = 0
    if (t>= 1) then
      S(j,(4-i)) = 1
      d = mod(d,2**i)
    end if
  end do
end do


do c = 1,7
  if (c == 1) then
    alpha = 0.5d0
  else if (c == 2) then
    alpha = 0.25d0
  else if (c == 3) then
    alpha = 0.1d0
  else if (c == 4) then
    alpha = 0.05d0
  else if (c == 5) then
    alpha = 0.025d0
```

```
else if (c == 6) then
  alpha = 0.01d0
else if (c == 7) then
  alpha = 0.005d0
end if

Subopt = 0d0
order = 0d0
SMALL = 1.0e+20
BIG = -1*(1.0e+20)

do y = 1,SIM

  Ropt = 0d0
  Rind = 0d0
  Reward1 = 0d0
  Reward2 = 0d0

  call GETrandom(seed,RANDOM)

  h = 0
  do i = 1,4
    do j = 1,4
      do k = 1,4
        do l = 1,4
          h = h + 1
          if (h == RANDOM) then
            STATE = (/i,j,k,l/)
          end if
        end do
      end do
    end do
  end do

  call GETPmatrix(seed,Pmat)
  call GETRmatrix(seed,Rmat)

    do b = 1,4
      do i = 1,4
        do j = 1,4
          P(i,j) = Pmat(b,i,j)
        end do
        r(i) = Rmat(b,i)
      end do
      h = 15
      G = 0
      IND = 0
      argmax = 0
      A = 0
      A(15,:) = (/1,1,1,1/)
      AV = 0
      do k = 1,E
        if (k > 1) call GETsubset(E,argmax,k,h)
          count = 0
          do i = 1,E
            count = count + S(h,i)
            countcomp = E - count
          end do
          if (k > 1) call GETAmatrix_LIMIT(AV,S,count,countcomp,P,h,A)
          call GETindices(b,k,S,h,r,A,argmax,G,IND,INDEX)
```

```
      end do
    end do

  INDEX2 = INDEX

  call Getbandit1(INDEX2,bandit1,cond)
  call GETrewards
  (STATE,Rmat,Pmat,INDEX2,alpha,speed1,speed2,bandit1,Ropt,Rind)

  Subopt(y) = Ropt - Rind

  if(Subopt(y) >= BIG) then
    BIG = Subopt(y)
  end if
  if(Subopt(y) <= SMALL) then
    SMALL = Subopt(y)
  end if

end do

ORDER = 1.0e+20

do j = 1,SIM
  do i = 1,SIM
    if(ORDER(j) >= Subopt(i)) then
      ORDER(j) = Subopt(i)
      k = i
    end if
  end do
  Subopt(k) = 1.0e+21
end do

j = 0
do i = 1,SIM
  if(ORDER(i) > 1.0e-7) j = j + 1
end do
if (mod(j,2) == 1 .and. j > 0)then
  MEDIAN = ORDER(SIM - j + ((j+1)/2))
else if (mod(j,2) == 0 .and. j > 0)then
  MEDIAN = (ORDER(SIM - j + (j/2)) + ORDER(SIM - j + ((j/2)+1)))/2d0
else if (j == 0) then
  MEDIAN = 0d0
end if

sum = 0d0
sum2 = 0d0
do i = 1,SIM
  sum = sum + ORDER(i)
  sum2 = sum2 + (ORDER(i))**2
end do

MEAN = sum/SIM*1d0
t = (((1d0/SIM*1d0)*sum2)) - (MEAN**2)
if(t > 0)then
  STD = sqrt((((1d0/SIM*1d0)*sum2)) - (MEAN**2))
else
  STD = 0d0
end if
PERCENT = 100.0d0 -(100d0*j/SIM*1d0)
```

```
  write(unit = 6, fmt= "(a)")"----------------------------------------"
  write(unit = 6, fmt= "(a,i2,a,i2,a,f6.4)")
  "speed1 = ",speed1," speed2 = " ,speed2," alpha = " ,alpha
  write(unit = 6, fmt= "(a)")"----------------------------------------"
  write(unit = 6, fmt= "(a,i3)")"     THE NUMBER OF SIMULATIONS IS  ", SIM
  write(unit = 6, fmt= "(a,f12.6,a,f12.6,a,f12.6)")
  "(1) Minimum", SMALL,"     (2) Median ",MEDIAN,"     (3) Maximum",BIG
  write(unit = 6, fmt= "(a,f12.6,a,f12.6,a,f12.6)")
  "(4) Mean   ", MEAN,"     (5) Std Dev",STD,"     (6) Percent",PERCENT
  write(unit = 6, fmt= "(a)")"========================================="

  open(unit = 7, file = "OPT_FINAL_5&2.dat")

  write(unit = type, fmt= "(a)")"----------------------------------------"
  write(unit = type, fmt= "(a,i2,a,i2,a,f6.4)")
  "speed1 = ",speed1," speed2 = " ,speed2," alpha = " ,alpha
  write(unit = type, fmt= "(a)")"----------------------------------------"
  write(unit = type, fmt= "(a,i3)")"     THE NUMBER OF SIMULATIONS IS  ",SIM
  write(unit = type, fmt= "(a,f12.6,a,f12.6,a,f12.6)")
  "(1) Minimum", SMALL,"     (2) Median ",MEDIAN,"    (3) Maximum",BIG
  write(unit = type, fmt= "(a,f12.6,a,f12.6,a,f12.6)")
  "(4) Mean   ", MEAN,"     (5) Std Dev",STD,"    (6) Percent",PERCENT
  write(unit = type, fmt= "(a)")"========================================="
end do

end program
!=====================================================================
subroutine GETsubset(E,argmax,k,h)
implicit none
integer :: k,h,E
integer, dimension(4) :: argmax

h = h - 2**(E - argmax(k - 1)) return
end
!=====================================================================
subroutine GETAmatrix_LIMIT(AV,S,count,countcomp,P,h,A)
implicit none
integer :: d,g,i,j,h,f,k,count,countcomp,E
double precision :: matsum,Z,sum
integer, dimension(15,4) :: S
double precision, dimension(4,4) :: P
double precision, dimension(15,4) :: A,AV
double precision, dimension(countcomp,countcomp) :: Mmat,Pmat
double precision, dimension(countcomp,1) :: Pvec,Psum,Product
double precision, dimension(count) :: V1,V2

E = 4
!-----------------------------------------------
Mmat = 0d0
!-----------------------------------------------
d = 0
do i = 1,E
  if(S(h,i) == 0) then
    if(d < countcomp) d = d + 1
    Mmat(d,d) = 1 - P(i,i)
    g = d
    do j = 1,E
      if (i < j) then
        if (S(h,j) == 0) then
          if (g < countcomp) g = g + 1
```

```
          Mmat(d,g) = (-1.0d0)*P(i,j)
          Mmat(g,d) = (-1.0d0)*P(j,i)
        end if
      end if
    end do
  end if
end do
!-------------------------------------------------
Psum = 0d0
Pvec = 0d0
Pmat = 0d0
d = 0
do i = 1,E
  g = 0
  if (S(h,i) == 0) then
    if (d < countcomp) d = d + 1
    do j = 1,E
      if (S(h,j) == 0) then
        if(g < countcomp) g = g + 1
        Pmat(d,g) = P(i,j)
      end if
    end do
  end if
end do
!-------------------------------------------------
do i = 1,countcomp
  do j = 1,countcomp
    Psum(i,1) = Psum(i,1) + Pmat(i,j)
  end do
end do
do i = 1,countcomp
  Pvec(i,1) = (1 - Psum(i,1))
end do
!-------------------------------------------------
Pvec = 1 + Pvec
!-------------------------------------------------
call GETinverse(countcomp,Mmat)
Product = 0d0
do i = 1,countcomp
  sum = 0d0
  do j = 1,countcomp
    sum = sum + (Mmat(i,j)*Pvec(j,1))
  end do
  Product(i,1) = sum
end do
Pvec = Product
!-------------------------------------------------
f = 0
do k = 1,4
  if (S(h,k) == 0) then
    if (f < countcomp) f = f + 1
    AV(h,k) = Pvec(f,1)
  end if
end do

d = 0
do i = 1,E
  matsum = 0
  if (S(h,i) == 1) then
    if (d < count) d = d + 1
```

```
    do j = 1,E
      if (S(h,j) == 0) matsum = matsum + (P(i,j) * AV(h,j))
    end do
    V1(d) = matsum
    matsum = 0
    do j = 1,E
      if (S(h,j) == 1) matsum = matsum + (P(i,j))
    end do
    V2(d) = matsum
    AV(h,i) = V1(d) + V2(d)
  end if
end do

AV(15,:) = (/1,1,1,1/)

f = 0
do k = 1,4
  if (S(h,k) == 1) then
    if (f < count) f = f + 1
    A(h,k) = AV(h,k)
  end if
end do
return
end
!=====================================================================
subroutine GETindices(b,k,S,h,r,A,argmax,G,IND,INDEX)
implicit none
integer :: b,i,h,k,l,j,E
double precision :: max,sum
integer, dimension(4) :: argmax
double precision, dimension(4) ::r,G,IND
double precision, dimension(15,4) :: A
integer, dimension(15,4) :: S
double precision, dimension(4,4) :: INDEX

if (k == 1) then
  max = -1000000
  do i = 1,4
    if (r(i)/A(h,i) > max) then
      max = r(i)/A(h,i)
      argmax(1) = i
    end if
  end do
  G(argmax(1)) = max
  IND(argmax(1)) = G(argmax(1))
else
  E = 4
  max = -1000000
  do i = 1,4
    l = 15
    sum = 0
    if (S(h,i) == 1) then
      do j = 1,k-1
        sum = sum + A(l,i)*G(argmax(j))
        l = l - 2**(E - argmax(j))
      end do
      if ((r(i) - sum)/A(h,i) > max) then
        max = (r(i) - sum)/A(h,i)
        argmax(k) = i
      end if
```

```
    end if
  end do
  G(argmax(k)) = max
  IND(argmax(k)) = G(argmax(k)) + IND(argmax(k - 1))
end if

INDEX(b,argmax(k)) = IND(argmax(k))
return
end
!=============================================================
subroutine GETinverse(countcomp,Mmat)
implicit none
integer :: i,j,countcomp
double precision :: a,b,c,d,determinant
double precision, dimension(countcomp,countcomp) :: Mmat
double precision, dimension(3,3) :: Cofactor

if (countcomp == 1) then
  Mmat(1,1) = 1d0/Mmat(1,1)
else if (countcomp == 2) then
  a = Mmat(1,1)
  b = Mmat(1,2)
  c = Mmat(2,1)
  d = Mmat(2,2)
  determinant = (a*d) - (b*c)
  Mmat(1,1) = d/determinant
  Mmat(1,2) = -1d0*(b/determinant)
  Mmat(2,1) = -1d0*(c/determinant)
  Mmat(2,2) = a/determinant
else if (countcomp == 3) then
  Cofactor(1,1) = ((Mmat(2,2)*Mmat(3,3)) - (Mmat(2,3)*Mmat(3,2)))
  Cofactor(1,2) = -1d0*((Mmat(2,1)*Mmat(3,3)) - (Mmat(2,3)*Mmat(3,1)))
  Cofactor(1,3) = ((Mmat(2,1)*Mmat(3,2)) - (Mmat(2,2)*Mmat(3,1)))

  Cofactor(2,1) = -1d0*((Mmat(1,2)*Mmat(3,3)) - (Mmat(1,3)*Mmat(3,2)))
  Cofactor(2,2) = ((Mmat(1,1)*Mmat(3,3)) - (Mmat(1,3)*Mmat(3,1)))
  Cofactor(2,3) = -1d0*((Mmat(1,1)*Mmat(3,2)) - (Mmat(1,2)*Mmat(3,1)))

  Cofactor(3,1) = ((Mmat(1,2)*Mmat(2,3)) - (Mmat(1,3)*Mmat(2,2)))
  Cofactor(3,2) = -1d0*((Mmat(1,1)*Mmat(2,3)) - (Mmat(1,3)*Mmat(2,1)))
  Cofactor(3,3) = ((Mmat(1,1)*Mmat(2,2)) - (Mmat(1,2)*Mmat(2,1)))

  determinant = Mmat(1,1)*Cofactor(1,1) + Mmat(1,2)*Cofactor(1,2)
              + Mmat(1,3)*Cofactor(1,3)

  do i = 1,3
    do j = 1,3
      Mmat(i,j) = Cofactor(j,i)/determinant
    end do
  end do
end if
return
end
!=============================================================
subroutine GETPmatrix(seed,Pmat)
implicit none
integer :: i,j,k
integer, dimension(3) :: seed
double precision ::UNIFORM,sum,L,U
double precision, dimension(4,4,4) :: PMat
```

```
L = 0.1
U = 0.9

  do k = 1,4
    do i = 1,4
      do j = 1,4
        call GETuniform(seed,L,U,UNIFORM)
        PMat(k,i,j) = UNIFORM
      end do
    end do
  end do

  do k = 1,4
    do i = 1,4
      sum = 0d0
      do j = 1,4
        sum = sum + PMat(k,i,j)
      end do
      do j = 1,4
        PMat(k,i,j) = Pmat(k,i,j)*(1.0d0/sum)
      end do
    end do
  end do
return end
!=================================================================
subroutine GETRmatrix(seed,Rmat)
implicit none
integer :: i,j
integer, dimension(3) :: seed
double precision :: L,U,x
double precision, dimension(4,4) :: RMat

L = 2d0
U = 5d0

do i = 1,4
  do j = 1,4
    call GETuniform(seed,L,U,x)
    RMat(i,j) = x
  end do
end do
return
end
!=================================================================
subroutine GETuniform(seed,L,U,x) implicit none integer,
dimension(3) :: seed double precision :: r,s,x,L,U

seed(1) = mod(171*seed(1),30269) seed(2) = mod(172*seed(2),30307)
seed(3) = mod(170*seed(3),30323)

s = seed(1)*1d0/30269 + seed(2)*1d0/30307 + seed(3)*1d0/30323 r =
s - int(s)

x = L + (U-L)*r

return end
!=================================================================
subroutine GETrandom(seed,x)
implicit none
```

```
integer :: x
integer, dimension(3) :: seed
double precision :: r,s

seed(1) = mod(171*seed(1),30269)
seed(2) = mod(172*seed(2),30307)
seed(3) = mod(170*seed(3),30323)

s = seed(1)*1d0/30269 + seed(2)*1d0/30307 + seed(3)*1d0/30323
r = s - int(s)

x = int(256*r)
return
end
!=====================================================================
subroutine Check(IND,num)
implicit none
integer :: i,j,k,c,d,num
double precision :: max
integer, dimension(4,4) :: ORDER1,ORDER2
double precision, dimension(4,4) :: IND

c = 0
d = 0
ORDER1 = 0
do k = 1,16
  max = -100000
  do i = 1,4
    do j = 1,4
      if (IND(i,j) > max) then
        max = IND(i,j)
        c = i
        d = j
      end if
    end do
  end do
  ORDER1(c,d) = k
  IND(c,d) = 0
end do

c = 0
d = 0
ORDER2 = 0
do k = 1,16
  max = -100000
  do i = 1,4
    do j = 1,4
      if (IND(i,j) > max) then
        max = IND(i,j)
        c = i
        d = j
      end if
    end do
  end do
  ORDER2(c,d) = k
  IND(c,d) = 0
end do

num = 0
do i = 1,4
```

```
  do j = 1,4
    if(ORDER1(i,j) /= ORDER2(i,j)) then
      num = num + 1
    end if
  end do
end do
return
end
!=====================================================================
subroutine GETbandit1(IND,bandit1,cond)
implicit none
integer :: i,j,c,cond,bandit1
double precision :: max
double precision, dimension(3) :: Gsub
double precision, dimension(4) :: min
double precision, dimension(4,4) :: IND

min = 0d0
do i= 1,4
  min(i) = 1000000d0
  do j = 1,4
    if (IND(i,j) < min(i)) then
      min(i) = IND(i,j)
    end if
  end do
end do

do i = 1,3
  max = -10000
  do j = 1,4
    if (min(j) > max) then
      max = min(j)
      Gsub(i) = min(j)
      c = j
    end if
  end do
  if (i == 1) then
    bandit1 = c
  end if
  min(c) = -100000
end do

if (Gsub(1) /= Gsub(2) .and. Gsub(2) /= Gsub(3) .and. Gsub(1) /= Gsub(3)) then
  cond = 1
else
  cond = 2
end if return end
!=====================================================================
subroutine GETrewards
(c,Rmat,Pmat,INDEX,alpha,speed1,speed2,bandit1,Rewoptimal,Rewpolicy)
implicit none integer ::
a,b,e,f,h,i,j,k,l,m,n,q,speed1,speed2,bandit1,bandit2,bplace
integer,dimension(3) :: v,uv,vu,cnew integer,dimension(4) ::
c,IS,u double precision :: alpha,Rmn,Rnm,maxi,Rewoptimal,Rewpolicy
double precision, dimension(2) :: dis double precision,
dimension(4) :: d double precision, dimension(6) :: max,rew double
precision, dimension(8) :: Psum double precision, dimension(4,2)
:: IP1,POL1 double precision, dimension(4,4) :: Rmat double
precision, dimension(4,4) :: INDEX double precision,
dimension(4,4,4) :: Pmat double precision, dimension(4,4,4,2) ::
```

```
IP2,POL2 double precision, dimension(4,4,4,4,2) :: O0,IP double
precision, dimension(4,4,4,4,4,2) :: O2,O4,O5,O6,O8 double
precision, dimension(4,4,4,4,4,4,2) :: O1,O3,O7,O9

d(1) = (1.0d0/alpha)*(1.0d0 - exp(-1.0d0*(alpha/(1.0d0*speed1))))
d(2) = (1.0d0/alpha)*(1.0d0 - exp(-1.0d0*(alpha/(1.0d0*speed2))))
d(3) = (1.0d0/alpha)*(1.0d0 - exp(-1.0d0*(alpha/(1.0d0*speed1*speed2))))
d(4) = exp(-1.0d0*(alpha/(1.0d0*speed1*speed2)))

dis(1) = exp(-1.0d0*(alpha/(1.0d0*speed1)))
dis(2) = exp(-1.0d0*(alpha/(1.0d0*speed2)))

O0=0d0
O1=0d0
O2=0d0
O3=0d0
O4=0d0
O5=0d0
O6=0d0
O7=0d0
O8=0d0
O9=0d0
IP1=0d0
IP2=0d0
IP=0d0
POL1 = 0d0
POL2 = 0d0

do q = 1,int((1.0d0*speed1*speed2)*(-1.0d0/alpha)
        *log((alpha*10e-8)/((5.0d0/d(1))+(5.0d0/d(2)))))+1
  if (mod(q,2) == 1) then
    e = 1
    f = 2
  else if (mod(q,2) == 0) then
    e = 2
    f = 1
  end if
  do i = 1,4
    do j = 1,4
      do k = 1,4
        do l = 1,4
          IS = (/i,j,k,l/)
          max(1) = -10000d0
          rew = 0d0
          do m = 1,4
            do a = 2,6
              max(a) = -10000d0
            end do
!--------------------------
            if (m == bandit1) then
              h = 0
              Psum(7) = 0d0
              do a = 1,4
                if (a /= bandit1) then
                  h = h + 1
                  uv(h) = IS(a)
                  vu(h) = a
                end if
                Psum(7) = Psum(7) + Pmat(m,IS(m),a)*IP1(a,e)
              end do
```

```
IP1(IS(bandit1),f) = Rmat(bandit1,IS(bandit1)) + dis(1)*Psum(7)
              maxi = -10000d0
              bandit2 = 0
              bplace = 0
              do b = 1,3
                if (INDEX(vu(b),uv(b)) > maxi) then
                  maxi = INDEX(vu(b),uv(b))
                  bandit2 = vu(b)
                  bplace = b
                end if
              end do
            end if
            h = 0
!----------------------------
            do n = 1,4
              if (m /= n) then
!----------------------------
                h = h + 1
!----------------------------
Rmn = ((d(3)*Rmat(m,IS(m))/d(1)) + (d(3)*Rmat(n,IS(n))/d(2)))
Rnm = ((Rmat(n,IS(n))/d(1)) + (Rmat(m,IS(m))/d(2)))*d(3)
                do b = 1,8
                  PSum(b) = 0d0
                end do
                do a = 1,4
!----------------------------
                  if (m == bandit1 .and. n == bandit2) then
                    v = uv
                    v(bplace) = a
Psum(8) = Psum(8) + Pmat(bandit2,IS(bandit2),a)*IP2(v(1),v(2),v(3),e)
                  end if
!----------------------------
                  u = IS
                  u(m) = a
Psum(1) = Psum(1) + Pmat(m,IS(m),a)*02(u(1),u(2),u(3),u(4),n,e)
Psum(2) = Psum(2) + Pmat(m,IS(m),a)*04(u(1),u(2),u(3),u(4),n,e)
Psum(3) = Psum(3) + Pmat(m,IS(m),a)*05(u(1),u(2),u(3),u(4),n,e)
Psum(4) = Psum(4) + Pmat(m,IS(m),a)*06(u(1),u(2),u(3),u(4),n,e)
Psum(5) = Psum(5) + Pmat(m,IS(m),a)*08(u(1),u(2),u(3),u(4),n,e)
                  do b = 1,4
                    u(n) = b
Psum(6) = Psum(6) + Pmat(n,IS(n),b)*Pmat(m,IS(m),a)*00(u(1),u(2),u(3),u(4),e)
                  end do
                end do
rew(1) = Rmn + d(4)*01(i,j,k,l,m,n,e)
rew(2) = Rnm + d(4)*03(i,j,k,l,n,m,e)
rew(3) = Rnm + d(4)*Psum(3)
rew(4) = Rmn + d(4)*Psum(4)
rew(5) = Rnm + d(4)*07(i,j,k,l,n,m,e)
rew(6) = Rnm + d(4)*09(i,j,k,l,n,m,e)
!-----------------------
                if (m == bandit1 .and. n == bandit2) then
IP2(uv(1),uv(2),uv(3),f) = Rmat(bandit2,IS(bandit2)) + dis(2)*Psum(8)
                end if
!-----------------------
                do a = 1,6
                  if (rew(a) > max(a)) then
                    max(a) = rew(a)
                  end if
                end do
```

```
O1(i,j,k,l,m,n,f) = Rmn + d(4)*Psum(1)
O3(i,j,k,l,m,n,f) = Rmn + d(4)*Psum(2)
O7(i,j,k,l,m,n,f) = Rmn + d(4)*Psum(5)
O9(i,j,k,l,m,n,f) = Rmn + d(4)*Psum(6)
              end if
            end do
O2(i,j,k,l,m,f) = max(2)
O4(i,j,k,l,m,f) = max(3)
O5(i,j,k,l,m,f) = max(4)
O6(i,j,k,l,m,f) = max(5)
O8(i,j,k,l,m,f) = max(6)
          end do
O0(i,j,k,l,f) = max(1)
        end do
      end do
    end do
  end do

  if(q == int((speed2*1d0)*(-1.0d0/alpha)
         *log((alpha*10e-8)/(5.0d0/d(2))))+1) then
  POL2 = IP2
  end if
  if(q == int((speed1*1d0)*(-1.0d0/alpha)
         *log((alpha*10e-8)/(5.0d0/d(1))))+1) then
  POL1 = IP1
  end if

end do Rewoptimal = O0(c(1),c(2),c(3),c(4),f)
h = 0
do a = 1,4
  if (a /= bandit1) then
    h = h + 1
    cnew(h) = c(a)
    v(h) = a
  end if
end do
Rewpolicy = POL1(c(bandit1),f) + POL2(cnew(1),cnew(2),cnew(3),f)

return end
!=======================================================================
```

Appendix D

```
program spd_opt_limit_4&3

implicit none
integer :: i,j,k,l,h,d,b,type,speed1,speed2,count,countcomp,E,M,SIZE
integer :: num,RANDOM,y,cond,c,bandit1,SIM
integer, dimension(3) :: seed
integer, dimension(4) :: argmax,STATE
double precision :: SMALL,BIG,MEAN,MEDIAN,STD,t,alpha,Z,sum,sum2,PERCENT
double precision :: Ropt,Rind,Reward1,Reward2,Reward3
double precision, dimension(4) :: r,G,IND
double precision, dimension(4,4) :: P,Rmat
integer, dimension(15,4) :: S
double precision, dimension(15,4) :: A,AV
double precision, dimension(4,4) :: INDEX,INDEX2
double precision, dimension(4,4,4) :: Pmat
double precision, dimension(500) :: Subopt,ORDER

type = 7
SIM = 500
speed1 = 4
speed2 = 3
seed = (/681,13754,21459/)
E = 4

do j = 15,1,-1
  d = j
  do i = 3,0,-1
    t = d/2**i
    if (t < 1) S(j,(4-i)) = 0
    if (t>= 1) then
      S(j,(4-i)) = 1
      d = mod(d,2**i)
    end if
  end do
end do

do c = 1,7
  if (c == 1) then
    alpha = 0.5d0
  else if (c == 2) then
    alpha = 0.25d0
  else if (c == 3) then
    alpha = 0.1d0
  else if (c == 4) then
    alpha = 0.05d0
  else if (c == 5) then
    alpha = 0.025d0
  else if (c == 6) then
```

```
  alpha = 0.01d0
else if (c == 7) then
  alpha = 0.005d0
end if

Subopt = 0d0
order = 0d0
SMALL = 1.0e+20
BIG = -1*(1.0e+20)

do y = 1,SIM
  Ropt = 0d0
  Rind = 0d0
  Reward1 = 0d0
  Reward2 = 0d0

  call GETrandom(seed,RANDOM)

  h = 0
  do i = 1,4
    do j = 1,4
      do k = 1,4
        do l = 1,4
          h = h + 1
          if (h == RANDOM) then
            STATE = (/i,j,k,l/)
          end if
        end do
      end do
    end do
  end do

  call GETPmatrix(seed,Pmat)
  call GETRmatrix(seed,Rmat)

    do b = 1,4
      do i = 1,4
        do j = 1,4
          P(i,j) = Pmat(b,i,j)
        end do
        r(i) = Rmat(b,i)
      end do
      h = 15
      G = 0
      IND = 0
      argmax = 0
      A = 0
      A(15,:) = (/1,1,1,1/)
      AV = 0
      do k = 1,E
        if (k > 1) call GETsubset(E,argmax,k,h)
          count = 0
          do i = 1,E
            count = count + S(h,i)
            countcomp = E - count
          end do
          if (k > 1) call GETAmatrix_LIMIT(AV,S,count,countcomp,P,h,A)
          call GETindices(b,k,S,h,r,A,argmax,G,IND,INDEX)
      end do
    end do
```

```
  INDEX2 = INDEX

  call Getbandit1(INDEX2,bandit1,cond)
  call GETrewards
  (STATE,Rmat,Pmat,INDEX2,alpha,speed1,speed2,bandit1,Ropt,Rind)

  Subopt(y) = Ropt - Rind

  if(Subopt(y) >= BIG) then
    BIG = Subopt(y)
  end if
  if(Subopt(y) <= SMALL) then
    SMALL = Subopt(y)
  end if

end do

ORDER = 1.0e+20

do j = 1,SIM
  do i = 1,SIM
    if(ORDER(j) >= Subopt(i)) then
      ORDER(j) = Subopt(i)
      k = i
    end if
  end do
  Subopt(k) = 1.0e+21
end do

j = 0
do i = 1,SIM
  if(ORDER(i) > 1.0e-7) j = j + 1
end do
if (mod(j,2) == 1 .and. j > 0)then
  MEDIAN = ORDER(SIM - j + ((j+1)/2))
else if (mod(j,2) == 0 .and. j > 0)then
  MEDIAN = (ORDER(SIM - j + (j/2)) + ORDER(SIM - j + ((j/2)+1)))/2d0
else if (j == 0) then
  MEDIAN = 0d0
end if

sum = 0d0
sum2 = 0d0
do i = 1,SIM
  sum = sum + ORDER(i)
  sum2 = sum2 + (ORDER(i))**2
end do

MEAN = sum/SIM*1d0
t = (((1d0/SIM*1d0)*sum2)) - (MEAN**2)
if(t > 0)then
  STD = sqrt((((1d0/SIM*1d0)*sum2)) - (MEAN**2))
else
  STD = 0d0
end if
PERCENT = 100.0d0 -(100d0*j/SIM*1d0)

write(unit = 6, fmt= "(a)")"----------------------------------------"
write(unit = 6, fmt= "(a,i2,a,i2,a,f6.4)")
```

```
  "speed1 = ",speed1," speed2 = " ,speed2," alpha = " ,alpha
  write(unit = 6, fmt= "(a)")"-----------------------------------------"
  write(unit = 6, fmt= "(a,i3)")"     THE NUMBER OF SIMULATIONS IS  ", SIM
  write(unit = 6, fmt= "(a,f12.6,a,f12.6,a,f12.6)")
  "(1) Minimum", SMALL,"     (2) Median ",MEDIAN,"     (3) Maximum",BIG
  write(unit = 6, fmt= "(a,f12.6,a,f12.6,a,f12.6)")
  "(4) Mean   ", MEAN,"     (5) Std Dev",STD,"     (6) Percent",PERCENT
  write(unit = 6, fmt= "(a)")"========================================="

  open(unit = 7, file = "OPT_FINAL.dat")

  write(unit = 7, fmt= "(a)")"-----------------------------------------"
  write(unit = 7, fmt= "(a,i2,a,i2,a,f6.4)")
  "speed1 = ",speed1," speed2 = " ,speed2," alpha = " ,alpha
  write(unit = 7, fmt= "(a)")"-----------------------------------------"
  write(unit = 7, fmt= "(a,i3)")"     THE NUMBER OF SIMULATIONS IS  ",SIM
  write(unit = 7, fmt= "(a,f12.6,a,f12.6,a,f12.6)")
  "(1) Minimum", SMALL,"     (2) Median ",MEDIAN,"    (3) Maximum",BIG
  write(unit = 7, fmt= "(a,f12.6,a,f12.6,a,f12.6)")
  "(4) Mean   ", MEAN,"     (5) Std Dev",STD,"    (6) Percent",PERCENT
  write(unit = 7, fmt= "(a)")"========================================="
end do

end program
!=====================================================================
subroutine GETsubset(E,argmax,k,h)
implicit none
integer :: k,h,E
integer, dimension(4) :: argmax

h = h - 2**(E - argmax(k - 1))
return
end
!=====================================================================
subroutine GETAmatrix_LIMIT(AV,S,count,countcomp,P,h,A)
implicit none
integer :: d,g,i,j,h,f,k,count,countcomp,E
double precision :: matsum,Z,sum
integer, dimension(15,4) :: S
double precision, dimension(4,4) :: P
double precision, dimension(15,4) :: A,AV
double precision, dimension(countcomp,countcomp) :: Mmat,Pmat
double precision, dimension(countcomp,1) :: Pvec,Psum,Product
double precision, dimension(count) :: V1,V2

E = 4
!-----------------------------------------------
Mmat = 0d0
!-----------------------------------------------
d = 0
do i = 1,E
  if(S(h,i) == 0) then
    if(d < countcomp) d = d + 1
    Mmat(d,d) = 1 - P(i,i)
    g = d
    do j = 1,E
      if (i < j) then
        if (S(h,j) == 0) then
          if (g < countcomp) g = g + 1
          Mmat(d,g) = (-1.0d0)*P(i,j)
```

```
          Mmat(g,d) = (-1.0d0)*P(j,i)
        end if
      end if
    end do
  end if
end do
!-------------------------------------------------
Psum = 0d0
Pvec = 0d0
Pmat = 0d0
d = 0
do i = 1,E
  g = 0
  if (S(h,i) == 0) then
    if (d < countcomp) d = d + 1
    do j = 1,E
      if (S(h,j) == 0) then
        if(g < countcomp) g = g + 1
        Pmat(d,g) = P(i,j)
      end if
    end do
  end if
end do
!-------------------------------------------------
do i = 1,countcomp
  do j = 1,countcomp
    Psum(i,1) = Psum(i,1) + Pmat(i,j)
  end do
end do
do i = 1,countcomp
  Pvec(i,1) = (1 - Psum(i,1))
end do
!-------------------------------------------------
Pvec = 1 + Pvec
!-------------------------------------------------
call GETinverse(countcomp,Mmat)
Product = 0d0
do i = 1,countcomp
  sum = 0d0
  do j = 1,countcomp
    sum = sum + (Mmat(i,j)*Pvec(j,1))
  end do
  Product(i,1) = sum
end do
Pvec = Product
!-------------------------------------------------
f = 0
do k = 1,4
  if (S(h,k) == 0) then
    if (f < countcomp) f = f + 1
    AV(h,k) = Pvec(f,1)
  end if
end do

d = 0
do i = 1,E
  matsum = 0
  if (S(h,i) == 1) then
    if (d < count) d = d + 1
    do j = 1,E
```

```
      if (S(h,j) == 0) matsum = matsum + (P(i,j) * AV(h,j))
    end do
    V1(d) = matsum
    matsum = 0
    do j = 1,E
      if (S(h,j) == 1) matsum = matsum + (P(i,j))
    end do
    V2(d) = matsum
    AV(h,i) = V1(d) + V2(d)
  end if
end do

AV(15,:) = (/1,1,1,1/)

f = 0
do k = 1,4
  if (S(h,k) == 1) then
    if (f < count) f = f + 1
    A(h,k) = AV(h,k)
  end if
end do
return
end
!=====================================================================
subroutine GETindices(b,k,S,h,r,A,argmax,G,IND,INDEX)
implicit none
integer :: b,i,h,k,l,j,E
double precision :: max,sum
integer, dimension(4) :: argmax
double precision, dimension(4) :: r,G,IND
double precision, dimension(15,4) :: A
integer, dimension(15,4) :: S
double precision, dimension(4,4) :: INDEX

if (k == 1) then
  max = -1000000
  do i = 1,4
    if (r(i)/A(h,i) > max) then
      max = r(i)/A(h,i)
      argmax(1) = i
    end if
  end do
  G(argmax(1)) = max
  IND(argmax(1)) = G(argmax(1))
else
  E = 4
  max = -1000000
  do i = 1,4
    l = 15
    sum = 0
    if (S(h,i) == 1) then
      do j = 1,k-1
        sum = sum + A(l,i)*G(argmax(j))
        l = l - 2**(E - argmax(j))
      end do
      if ((r(i) - sum)/A(h,i) > max) then
        max = (r(i) - sum)/A(h,i)
        argmax(k) = i
      end if
    end if
```

```
  end do
  G(argmax(k)) = max
  IND(argmax(k)) = G(argmax(k)) + IND(argmax(k - 1))
end if

INDEX(b,argmax(k)) = IND(argmax(k))
return
end
!=============================================================
subroutine GETinverse(countcomp,Mmat)
implicit none
integer :: i,j,countcomp
double precision :: a,b,c,d,determinant
double precision, dimension(countcomp,countcomp) :: Mmat
double precision, dimension(3,3) :: Cofactor

if (countcomp == 1) then
  Mmat(1,1) = 1d0/Mmat(1,1)
else if (countcomp == 2) then
  a = Mmat(1,1)
  b = Mmat(1,2)
  c = Mmat(2,1)
  d = Mmat(2,2)
  determinant = (a*d) - (b*c)
  Mmat(1,1) = d/determinant
  Mmat(1,2) = -1d0*(b/determinant)
  Mmat(2,1) = -1d0*(c/determinant)
  Mmat(2,2) = a/determinant
else if (countcomp == 3) then
  Cofactor(1,1) = ((Mmat(2,2)*Mmat(3,3)) - (Mmat(2,3)*Mmat(3,2)))
  Cofactor(1,2) = -1d0*((Mmat(2,1)*Mmat(3,3)) - (Mmat(2,3)*Mmat(3,1)))
  Cofactor(1,3) = ((Mmat(2,1)*Mmat(3,2)) - (Mmat(2,2)*Mmat(3,1)))

  Cofactor(2,1) = -1d0*((Mmat(1,2)*Mmat(3,3)) - (Mmat(1,3)*Mmat(3,2)))
  Cofactor(2,2) = ((Mmat(1,1)*Mmat(3,3)) - (Mmat(1,3)*Mmat(3,1)))
  Cofactor(2,3) = -1d0*((Mmat(1,1)*Mmat(3,2)) - (Mmat(1,2)*Mmat(3,1)))

  Cofactor(3,1) = ((Mmat(1,2)*Mmat(2,3)) - (Mmat(1,3)*Mmat(2,2)))
  Cofactor(3,2) = -1d0*((Mmat(1,1)*Mmat(2,3)) - (Mmat(1,3)*Mmat(2,1)))
  Cofactor(3,3) = ((Mmat(1,1)*Mmat(2,2)) - (Mmat(1,2)*Mmat(2,1)))

  determinant = Mmat(1,1)*Cofactor(1,1) + Mmat(1,2)*Cofactor(1,2)
              + Mmat(1,3)*Cofactor(1,3)
  do i = 1,3
    do j = 1,3
      Mmat(i,j) = Cofactor(j,i)/determinant
    end do
  end do
end if
return
end
!=============================================================
subroutine GETPmatrix(seed,Pmat)
implicit none
integer :: i,j,k
integer, dimension(3) :: seed
double precision ::UNIFORM,sum,L,U
double precision, dimension(4,4,4) :: PMat

L = 0.1
```

```
U = 0.9

  do k = 1,4
    do i = 1,4
      do j = 1,4
        call GETuniform(seed,L,U,UNIFORM)
        PMat(k,i,j) = UNIFORM
      end do
    end do
  end do

  do k = 1,4
    do i = 1,4
      sum = 0d0
      do j = 1,4
        sum = sum + PMat(k,i,j)
      end do
      do j = 1,4
        PMat(k,i,j) = Pmat(k,i,j)*(1.0d0/sum)
      end do
    end do
  end do
return
end
!=============================================================
subroutine GETRmatrix(seed,Rmat)
implicit none
integer :: i,j
integer, dimension(3) :: seed
double precision :: L,U,x
double precision, dimension(4,4) :: RMat

L = 2d0
U = 5d0

do i = 1,4
  do j = 1,4
    call GETuniform(seed,L,U,x)
    RMat(i,j) = x
  end do
end do
return
end
!=============================================================
subroutine GETuniform(seed,L,U,x)
implicit none
integer, dimension(3) :: seed
double precision :: r,s,x,L,U

seed(1) = mod(171*seed(1),30269)
seed(2) = mod(172*seed(2),30307)
seed(3) = mod(170*seed(3),30323)

s = seed(1)*1d0/30269 + seed(2)*1d0/30307 + seed(3)*1d0/30323
r = s - int(s)

x = L + (U-L)*r
return
end
!=============================================================
```

```
subroutine GETrandom(seed,x) implicit none integer :: x integer,
dimension(3) :: seed double precision :: r,s

seed(1) = mod(171*seed(1),30269) seed(2) = mod(172*seed(2),30307)
seed(3) = mod(170*seed(3),30323)

s = seed(1)*1d0/30269 + seed(2)*1d0/30307 + seed(3)*1d0/30323 r =
s - int(s)

x = int(256*r) return end
!=====================================================================
subroutine Check(IND,num)
implicit none
integer :: i,j,k,c,d,num
double precision :: max
integer, dimension(4,4) :: ORDER1,ORDER2
double precision, dimension(4,4) :: IND

c = 0
d = 0
ORDER1 = 0
do k = 1,16
  max = -100000
  do i = 1,4
    do j = 1,4
      if (IND(i,j) > max) then
        max = IND(i,j)
        c = i
        d = j
      end if
    end do
  end do
  ORDER1(c,d) = k
  IND(c,d) = 0
end do

c = 0
d = 0
ORDER2 = 0
do k = 1,16
  max = -100000
  do i = 1,4
    do j = 1,4
      if (IND(i,j) > max) then
        max = IND(i,j)
        c = i
        d = j
      end if
    end do
  end do
  ORDER2(c,d) = k
  IND(c,d) = 0
end do

num = 0
do i = 1,4
  do j = 1,4
    if(ORDER1(i,j) /= ORDER2(i,j)) then
      num = num + 1
    end if
```

```
  end do
end do
return
end
!=====================================================================
subroutine GETbandit1(IND,bandit1,cond)
implicit none
integer :: i,j,c,cond,bandit1
double precision :: max
double precision, dimension(3) :: Gsub
double precision, dimension(4) :: min
double precision, dimension(4,4) :: IND

min = 0d0
do i= 1,4
  min(i) = 1000000d0
  do j = 1,4
    if (IND(i,j) < min(i)) then
      min(i) = IND(i,j)
    end if
  end do
end do

do i = 1,3
  max = -10000
  do j = 1,4
    if (min(j) > max) then
      max = min(j)
      Gsub(i) = min(j)
      c = j
    end if
  end do
  if (i == 1) then
    bandit1 = c
  end if
  min(c) = -100000
end do

if (Gsub(1) /= Gsub(2) .and. Gsub(2) /= Gsub(3) .and. Gsub(1) /= Gsub(3)) then
  cond = 1
else
  cond = 2
end if
return
end
!=====================================================================
subroutine GETrewards
(c,Rmat,Pmat,INDEX,alpha,speed1,speed2,bandit1,Rewoptimal,Rewpolicy)
implicit none
integer :: a,b,e,f,h,i,j,k,l,m,n,q,speed1,speed2,bandit1,bandit2,bplace
integer,dimension(3) :: v,uv,vu,cnew
integer,dimension(4) :: c,IS,u
double precision :: alpha,Rmn,Rnm,maxi,Rewoptimal,Rewpolicy
double precision, dimension(2) :: dis
double precision, dimension(4) :: d
double precision, dimension(6) :: max,rew
double precision, dimension(8) :: Psum
double precision, dimension(4,2) :: IP1,POL1
double precision, dimension(4,4) :: Rmat
double precision, dimension(4,4) :: INDEX
```

```
double precision, dimension(4,4,4) :: Pmat
double precision, dimension(4,4,4,2) :: IP2,POL2
double precision, dimension(4,4,4,4,2) :: O0,IP
double precision, dimension(4,4,4,4,4,2) :: O3,O4,O6,O8,O9
double precision, dimension(4,4,4,4,4,4,2) :: O1,O2,O5,O7,O10,O11

d(1) = (1.0d0/alpha)*(1.0d0 - exp(-1.0d0*(alpha/(1.0d0*speed1))))
d(2) = (1.0d0/alpha)*(1.0d0 - exp(-1.0d0*(alpha/(1.0d0*speed2))))
d(3) = (1.0d0/alpha)*(1.0d0 - exp(-1.0d0*(alpha/12.0d0)))
d(4) = exp(-1.0d0*(alpha/12.0d0))

dis(1) = exp(-1.0d0*(alpha/(1.0d0*speed1)))
dis(2) = exp(-1.0d0*(alpha/(1.0d0*speed2)))

O0=0d0
O1=0d0
O2=0d0
O3=0d0
O4=0d0
O5=0d0
O6=0d0
O7=0d0
O8=0d0
O9=0d0
O10=0d0
O11=0d0
IP1=0d0
IP2=0d0
IP=0d0
POL1 = 0d0
POL2 = 0d0

do q = 1,int((1.0d0*speed1*speed2)*(-1.0d0/alpha)
        *log((alpha*10e-8)/((5.0d0/d(1))+(5.0d0/d(2)))))+1
  if (mod(q,2) == 1) then
    e = 1
    f = 2
  else if (mod(q,2) == 0) then
    e = 2
    f = 1
  end if
  do i = 1,4
    do j = 1,4
      do k = 1,4
        do l = 1,4
          IS = (/i,j,k,l/)
          max(1) = -10000d0
          rew = 0d0
          do m = 1,4
            do a = 2,6
              max(a) = -10000d0
            end do
!-------------------------
            if (m == bandit1) then
              h = 0
              Psum(7) = 0d0
              do a = 1,4
                if (a /= bandit1) then
                  h = h + 1
                  uv(h) = IS(a)
```

```
                vu(h) = a
              end if
              Psum(7) = Psum(7) + Pmat(m,IS(m),a)*IP1(a,e)
            end do
IP1(IS(bandit1),f) = Rmat(bandit1,IS(bandit1)) + dis(1)*Psum(7)
            maxi = -10000d0
            bandit2 = 0
            bplace = 0
            do b = 1,3
              if (INDEX(vu(b),uv(b)) > maxi) then
                maxi = INDEX(vu(b),uv(b))
                bandit2 = vu(b)
                bplace = b
              end if
            end do
          end if
          h = 0
!---------------------------
          do n = 1,4
            if (m /= n) then
!---------------------------
              h = h + 1
!---------------------------
Rmn = ((d(3)*Rmat(m,IS(m))/d(1)) + (d(3)*Rmat(n,IS(n))/d(2)))
Rnm = ((Rmat(n,IS(n))/d(1)) + (Rmat(m,IS(m))/d(2)))*d(3)
              do b = 1,8
                PSum(b) = 0d0
              end do
              do a = 1,4
!---------------------------
                if (m == bandit1 .and. n == bandit2) then
                  v = uv
                  v(bplace) = a
Psum(8) = Psum(8) + Pmat(bandit2,IS(bandit2),a)*IP2(v(1),v(2),v(3),e)
                end if
!---------------------------
                u = IS
                u(m) = a
Psum(1) = Psum(1) + Pmat(m,IS(m),a)*03(u(1),u(2),u(3),u(4),n,e)
Psum(2) = Psum(2) + Pmat(m,IS(m),a)*04(u(1),u(2),u(3),u(4),n,e)
Psum(3) = Psum(3) + Pmat(m,IS(m),a)*06(u(1),u(2),u(3),u(4),n,e)
Psum(4) = Psum(4) + Pmat(m,IS(m),a)*08(u(1),u(2),u(3),u(4),n,e)
Psum(5) = Psum(5) + Pmat(m,IS(m),a)*09(u(1),u(2),u(3),u(4),n,e)
                do b = 1,4
                  u(n) = b
Psum(6) = Psum(6) + Pmat(n,IS(n),b)*Pmat(m,IS(m),a)*00(u(1),u(2),u(3),u(4),e)
                end do
              end do
rew(1) = Rmn + d(4)*01(i,j,k,l,m,n,e)
rew(2) = Rnm + d(4)*Psum(2)
rew(3) = Rmn + d(4)*05(i,j,k,l,m,n,e)
rew(4) = Rnm + d(4)*07(i,j,k,l,n,m,e)
rew(5) = Rmn + d(4)*Psum(5)
rew(6) = Rnm + d(4)*010(i,j,k,l,n,m,e)
!-----------------------
              if (m == bandit1 .and. n == bandit2) then
IP2(uv(1),uv(2),uv(3),f) = Rmat(bandit2,IS(bandit2)) + dis(2)*Psum(8)
              end if
!-----------------------
              do a = 1,6
```

```
                  if (rew(a) > max(a)) then
                    max(a) = rew(a)
                  end if
                end do
O1(i,j,k,l,m,n,f) = Rmn + d(4)*O2(i,j,k,l,m,n,e)
O2(i,j,k,l,m,n,f) = Rmn + d(4)*Psum(1)
O5(i,j,k,l,m,n,f) = Rmn + d(4)*Psum(3)
O7(i,j,k,l,n,m,f) = Rnm + d(4)*Psum(4)
O10(i,j,k,l,m,n,f) = Rmn + d(4)*O11(i,j,k,l,m,n,e)
O11(i,j,k,l,m,n,f) = Rmn + d(4)*Psum(6)
              end if
            end do
O3(i,j,k,l,m,f) = max(2)
O4(i,j,k,l,m,f) = max(3)
O6(i,j,k,l,m,f) = max(4)
O8(i,j,k,l,m,f) = max(5)
O9(i,j,k,l,m,f) = max(6)
          end do
O0(i,j,k,l,f) = max(1)
        end do
      end do
    end do
  end do

  if(q == int((speed2*1d0)*(-1.0d0/alpha)
        *log((alpha*10e-8)/(5.0d0/d(2))))+1) then
  POL2 = IP2
  end if
  if(q == int((speed1*1d0)*(-1.0d0/alpha)
        *log((alpha*10e-8)/(5.0d0/d(1))))+1) then
  POL1 = IP1
  end if

end do
Rewoptimal = O0(c(1),c(2),c(3),c(4),f)
h = 0
do a = 1,4
  if (a /= bandit1) then
    h = h + 1
    cnew(h) = c(a)
    v(h) = a
  end if
end do Rewpolicy = POL1(c(bandit1),f) +
POL2(cnew(1),cnew(2),cnew(3),f)
return
end
!=============================================================================
```

Appendix E

```
program spd_opt_limit_vsGittins_6&1

implicit none
integer :: i,j,k,l,h,d,b,type,speed1,speed2,count,countcomp,E,M,SIZE
integer :: num,RANDOM,y,cond,c,bandit1,SIM
integer, dimension(3) :: seed
integer, dimension(4) :: argmax,STATE
double precision :: SMALL,BIG,MEAN,MEDIAN,STD,t,alpha,Z,sum,sum2,PERCENT
double precision :: SMALL2,BIG2,MEAN2,MEDIAN2,STD2,PERCENT2,t1,t2
double precision :: SMALL3,BIG3,MEAN3,MEDIAN3,STD3,PERCENT3,t3,t4
double precision :: Ropt,Rind,Reward1,Reward2,Reward3,Rgit
double precision, dimension(4) :: r,G,IND
double precision, dimension(4,4) :: P,Rmat
integer, dimension(15,4) :: S
double precision, dimension(15,4) :: A,AV
double precision, dimension(4,4) :: INDEX,INDEX2
double precision, dimension(4,4,4) :: Pmat
double precision, dimension(1000) :: Subopt,ORDER,Subopt2,ORDER2,Subopt3,ORDER3

type = 7
SIM = 1000
speed1 = 6
speed2 = 1
seed = (/16598,12498,568/)
E = 4

do j = 15,1,-1
  d = j
  do i = 3,0,-1
    t = d/2**i
    if (t < 1) S(j,(4-i)) = 0
    if (t>= 1) then
      S(j,(4-i)) = 1
      d = mod(d,2**i)
    end if
  end do
end do

do c = 1,7
  if (c == 1) then
    alpha = 0.5d0
  else if (c == 2) then
    alpha = 0.25d0
  else if (c == 3) then
    alpha = 0.1d0
  else if (c == 4) then
    alpha = 0.05d0
  else if (c == 5) then
```

```
  alpha = 0.025d0
else if (c == 6) then
  alpha = 0.01d0
else if (c == 7) then
  alpha = 0.005d0
end if

Subopt = 0d0
Subopt2 = 0d0
Subopt3 = 0d0
order = 0d0
order2 = 0d0
order3 = 0d0
SMALL = 1.0e+20
BIG = -1*(1.0e+20)
SMALL2 = 1.0e+20
BIG2 = -1*(1.0e+20)
SMALL3 = 1.0e+20
BIG3 = -1*(1.0e+20)

do y = 1,SIM

  Ropt = 0d0
  Rind = 0d0
  Rgit = 0d0
  Reward1 = 0d0
  Reward2 = 0d0

  call GETrandom(seed,RANDOM)

  h = 0
  do i = 1,4
    do j = 1,4
      h = h + 1
      if (h == RANDOM) then
        STATE = (/1,1,i,j/)
      end if
    end do
  end do

  call GETPmatrix(seed,Pmat)
  call GETRmatrix(seed,Rmat)

    do b = 1,4
      do i = 1,4
        do j = 1,4
          P(i,j) = Pmat(b,i,j)
        end do
        r(i) = Rmat(b,i)
      end do
      h = 15
      G = 0
      IND = 0
      argmax = 0
      A = 0
      A(15,:) = (/1,1,1,1/)
      AV = 0
      do k = 1,E
        if (k > 1) call GETsubset(E,argmax,k,h)
          count = 0
```

```
            do i = 1,E
              count = count + S(h,i)
              countcomp = E - count
            end do
            if (k > 1) call GETAmatrix_LIMIT(AV,S,count,countcomp,P,h,A)
            call GETindices(b,k,S,h,r,A,argmax,G,IND,INDEX)
        end do
      end do

    INDEX2 = INDEX

    call Getbandit1(INDEX,bandit1,cond)
    call GETrewards
    (STATE,Rmat,Pmat,INDEX2,alpha,speed1,speed2,bandit1,Ropt,Rind,Rgit)

    Subopt(y) = Ropt - Rind
    Subopt2(y) = Ropt - Rgit
    Subopt3(y) = Rind - Rgit

    if(Subopt(y) >= BIG) then
      BIG = Subopt(y)
    end if
    if(Subopt(y) <= SMALL) then
      SMALL = Subopt(y)
    end if
    if(Subopt2(y) >= BIG2) then
      BIG2 = Subopt2(y)
    end if
    if(Subopt2(y) <= SMALL2) then
      SMALL2 = Subopt2(y)
    end if
    if(Subopt3(y) >= BIG3) then
      BIG3 = Subopt3(y)
    end if
    if(Subopt3(y) <= SMALL3) then
      SMALL3 = Subopt3(y)
    end if

end do

ORDER = 1.0e+20

do j = 1,SIM
  do i = 1,SIM
    if(ORDER(j) >= Subopt(i)) then
      ORDER(j) = Subopt(i)
      k = i
    end if
  end do
  Subopt(k) = 1.0e+21
end do

j = 0
do i = 1,SIM
  if(ORDER(i) > 10e-7) j = j + 1
end do
if (mod(j,2) == 1 .and. j > 0)then
  MEDIAN = ORDER(SIM - j + ((j+1)/2))
else if (mod(j,2) == 0 .and. j > 0)then
  MEDIAN = (ORDER(SIM - j + (j/2)) + ORDER(SIM - j + ((j/2)+1)))/2d0
```

```
  else if (j == 0) then
    MEDIAN = 0d0
  end if

  sum = 0d0
  sum2 = 0d0
  do i = 1,SIM
    sum = sum + ORDER(i)
    sum2 = sum2 + (ORDER(i))**2
  end do

  MEAN = sum/SIM*1d0
  t = (((1d0/SIM*1d0)*sum2)) - (MEAN**2)
  if(t > 0)then
    STD = sqrt((((1d0/SIM*1d0)*sum2)) - (MEAN**2))
  else
    STD = 0d0
  end if
  PERCENT = 100.0d0 -(100d0*j/SIM*1d0)
!-----------------------------------------------------------------------
 write(unit = 6, fmt= "(a,f6.4)")"alpha = " ,alpha
 write(unit = 6, fmt= "(a)")"STATISTICS FOR Roptimal - Rpolicy "
 write(unit = 6, fmt= "(a,f12.6,a,f12.6,a,f12.6)")
 "(1) Minimum", SMALL,"    (2) Median ",MEDIAN,"     (3) Maximum",BIG
 write(unit = 6, fmt= "(a,f12.6,a,f12.6,a,f12.6)")
 "(4) Mean   ", MEAN,"    (5) Std Dev",STD,"     (6) Percent",PERCENT

 open(unit = 7, file = "US_vs_GITTINS_6&1.dat")
 write(unit = 7, fmt= "(a,f6.4)")"alpha = " ,alpha

 write(unit = 7e, fmt= "(a)")"STATISTICS FOR Roptimal - Rpolicy "
 write(unit = 7, fmt= "(a,f12.6,a,f12.6,a,f12.6)")
 "(1) Minimum", SMALL,"    (2) Median ",MEDIAN,"     (3) Maximum",BIG
 write(unit = 7, fmt= "(a,f12.6,a,f12.6,a,f12.6)")
 "(4) Mean   ", MEAN,"    (5) Std Dev",STD,"     (6) Percent",PERCENT
!-----------------------------------------------------------------------
  ORDER2 = 1.0e+20
  do j = 1,SIM
    do i = 1,SIM
      if(ORDER2(j) >= Subopt2(i)) then
        ORDER2(j) = Subopt2(i)
       k = i
      end if
   end do
    Subopt2(k) = 1.0e+21
  end do

  j = 0
  do i = 1,SIM
    if(ORDER2(i) > 10e-7) j = j + 1
  end do
  if (mod(j,2) == 1 .and. j > 0)then
    MEDIAN2 = ORDER2(SIM - j + ((j+1)/2))
  else if (mod(j,2) == 0 .and. j > 0)then
    MEDIAN2 = (ORDER2(SIM - j + (j/2)) + ORDER2(SIM - j + ((j/2)+1)))/2d0
  else if (j == 0) then
    MEDIAN2 = 0d0
  end if

  sum = 0d0
```

```
  sum2 = 0d0
  do i = 1,SIM
    sum = sum + ORDER2(i)
    sum2 = sum2 + ((ORDER2(i))**2)
  end do

  MEAN2 = (sum/SIM*1d0)
  t2 = (((1d0/SIM*1d0)*sum2)) - (MEAN2**2)

  if(t2 > 0)then
    STD2 = sqrt((((1d0/SIM*1d0)*sum2)) - ((MEAN2)**2))
  else
    STD2 = 0d0
  end if
  PERCENT2 = 100.0d0 - (100.0d0*j/SIM*1.0d0)
!-----------------------------------------------------------------------
write(unit = 6, fmt= "(a)")"STATISTICS FOR Roptimal - Rgittins "
write(unit = 6, fmt= "(a,f12.6,a,f12.6,a,f12.6)") "(1) Minimum",
SMALL2,"    (2) Median ",MEDIAN2,"     (3) Maximum",BIG2
write(unit = 6, fmt= "(a,f12.6,a,f12.6,a,f12.6)") "(4) Mean   ",
MEAN2,"    (5) Std Dev",STD2,"     (6) Percent",PERCENT2

write(unit = 7, fmt= "(a)")"STATISTICS FOR Roptimal - Rgittins "
write(unit = 7, fmt= "(a,f12.6,a,f12.6,a,f12.6)") "(1) Minimum",
SMALL2,"    (2) Median ",MEDIAN2,"     (3) Maximum",BIG2
write(unit = 7, fmt= "(a,f12.6,a,f12.6,a,f12.6)") "(4) Mean ",
MEAN2,"    (5) Std Dev",STD2,"     (6) Percent",PERCENT2
!-----------------------------------------------------------------------
  ORDER3 = 1.0e+20
  do j = 1,SIM
    do i = 1,SIM
      if(ORDER3(j) >= Subopt3(i)) then
        ORDER3(j) = Subopt3(i)
       k = i
      end if
   end do
    Subopt3(k) = 1.0e+21
  end do

  j = 0
  do i = 1,SIM
    if(ORDER3(i) > 10e-7) j = j + 1
  end do
  if (mod(j,2) == 1 .and. j > 0)then
    MEDIAN3 = ORDER3(SIM - j + ((j+1)/2))
  else if (mod(j,2) == 0 .and. j > 0)then
    MEDIAN3 = (ORDER3(SIM - j + (j/2)) + ORDER3(SIM - j + ((j/2)+1)))/2d0
  else if (j == 0) then
    MEDIAN3 = 0d0
  end if

  sum = 0d0
  sum2 = 0d0
  do i = 1,SIM
    sum = sum + ORDER3(i)
    sum2 = sum2 + ((ORDER3(i))**2)
  end do

  MEAN3 = (sum/SIM*1d0)
  t3 = (((1d0/SIM*1d0)*sum2)) - (MEAN3**2)
```

```
  if(t3 > 0)then
    STD3 = sqrt((((1d0/SIM*1d0)*sum2)) - ((MEAN3)**2))
  else
    STD3 = 0d0
  end if
  PERCENT3 = 100.0d0 - (100.0d0*j/SIM*1.0d0)
!-----------------------------------------------------------------------
write(unit = 6, fmt= "(a)")"STATISTICS FOR Rpolicy - Rgittins "
write(unit = 6, fmt= "(a,f12.6,a,f12.6,a,f12.6)") "(1) Minimum",
SMALL3,"    (2) Median ",MEDIAN3,"     (3) Maximum",BIG3
write(unit = 6, fmt= "(a,f12.6,a,f12.6,a,f12.6)") "(4) Mean   ",
MEAN3,"    (5) Std Dev",STD3,"     (6) Percent",PERCENT3

write(unit = 7, fmt= "(a)")"STATISTICS FOR Rpolicy - Rgittins "
write(unit = 7, fmt= "(a,f12.6,a,f12.6,a,f12.6)") "(1) Minimum",
SMALL3,"    (2) Median ",MEDIAN3,"     (3) Maximum",BIG3
write(unit = 7, fmt= "(a,f12.6,a,f12.6,a,f12.6)") "(4) Mean ",
MEAN3,"    (5) Std Dev",STD3,"     (6) Percent",PERCENT3
!-----------------------------------------------------------------------
end do
!=======================================================================
subroutine GETsubset(E,argmax,k,h) implicit none integer :: k,h,E
integer, dimension(4) :: argmax

h = h - 2**(E - argmax(k - 1))
return
end
!=======================================================================
subroutine GETAmatrix_LIMIT(AV,S,count,countcomp,P,h,A)
implicit none
integer :: d,g,i,j,h,f,k,count,countcomp,E
double precision :: matsum,Z,sum
integer, dimension(15,4) :: S
double precision, dimension(4,4) :: P
double precision, dimension(15,4) :: A,AV
double precision, dimension(countcomp,countcomp) :: Mmat,Pmat
double precision, dimension(countcomp,1) :: Pvec,Psum,Product
double precision, dimension(count) :: V1,V2

E = 4
!-------------------------------------------------
Mmat = 0d0
!-------------------------------------------------
d = 0
do i = 1,E
  if(S(h,i) == 0) then
    if(d < countcomp) d = d + 1
    Mmat(d,d) = 1 - P(i,i)
    g = d
    do j = 1,E
      if (i < j) then
        if (S(h,j) == 0) then
          if (g < countcomp) g = g + 1
          Mmat(d,g) = (-1.0d0)*P(i,j)
          Mmat(g,d) = (-1.0d0)*P(j,i)
        end if
      end if
    end do
  end if
```

```
end do
!-------------------------------------------------
Psum = 0d0
Pvec = 0d0
Pmat = 0d0
d = 0
do i = 1,E
  g = 0
  if (S(h,i) == 0) then
    if (d < countcomp) d = d + 1
    do j = 1,E
      if (S(h,j) == 0) then
        if(g < countcomp) g = g + 1
        Pmat(d,g) = P(i,j)
      end if
    end do
  end if
end do
!-------------------------------------------------
do i = 1,countcomp
  do j = 1,countcomp
    Psum(i,1) = Psum(i,1) + Pmat(i,j)
  end do
end do
do i = 1,countcomp
  Pvec(i,1) = (1 - Psum(i,1))
end do
!-------------------------------------------------
Pvec = 1 + Pvec
!-------------------------------------------------
call GETinverse(countcomp,Mmat)
Product = 0d0
do i = 1,countcomp
  sum = 0d0
  do j = 1,countcomp
    sum = sum + (Mmat(i,j)*Pvec(j,1))
  end do
  Product(i,1) = sum
end do
Pvec = Product
!-------------------------------------------------
f = 0
do k = 1,4
  if (S(h,k) == 0) then
    if (f < countcomp) f = f + 1
    AV(h,k) = Pvec(f,1)
  end if
end do

d = 0
do i = 1,E
  matsum = 0
  if (S(h,i) == 1) then
    if (d < count) d = d + 1
    do j = 1,E
      if (S(h,j) == 0) matsum = matsum + (P(i,j) * AV(h,j))
    end do
    V1(d) = matsum
    matsum = 0
    do j = 1,E
```

```
      if (S(h,j) == 1) matsum = matsum + (P(i,j))
    end do
    V2(d) = matsum
    AV(h,i) = V1(d) + V2(d)
  end if
end do

AV(15,:) = (/1,1,1,1/)

f = 0
do k = 1,4
  if (S(h,k) == 1) then
    if (f < count) f = f + 1
    A(h,k) = AV(h,k)
  end if
end do
return
end
!=====================================================================
subroutine GETindices(b,k,S,h,r,A,argmax,G,IND,INDEX)
implicit none
integer :: b,i,h,k,l,j,E
double precision :: max,sum
integer, dimension(4) :: argmax
double precision, dimension(4) :: r,G,IND
double precision, dimension(15,4) :: A
integer, dimension(15,4) :: S
double precision, dimension(4,4) :: INDEX

if (k == 1) then
  max = -1000000
  do i = 1,4
    if (r(i)/A(h,i) > max) then
      max = r(i)/A(h,i)
      argmax(1) = i
    end if
  end do
  G(argmax(1)) = max
  IND(argmax(1)) = G(argmax(1))
else
  E = 4
  max = -1000000
  do i = 1,4
    l = 15
    sum = 0
    if (S(h,i) == 1) then
      do j = 1,k-1
        sum = sum + A(l,i)*G(argmax(j))
        l = l - 2**(E - argmax(j))
      end do
      if ((r(i) - sum)/A(h,i) > max) then
        max = (r(i) - sum)/A(h,i)
        argmax(k) = i
      end if
    end if
  end do
  G(argmax(k)) = max
  IND(argmax(k)) = G(argmax(k)) + IND(argmax(k - 1))
end if
```

```
INDEX(b,argmax(k)) = IND(argmax(k))
return
end
!=============================================================
subroutine GETinverse(countcomp,Mmat)
implicit none
integer :: i,j,countcomp
double precision :: a,b,c,d,determinant
double precision, dimension(countcomp,countcomp) :: Mmat
double precision, dimension(3,3) :: Cofactor

if (countcomp == 1) then
  Mmat(1,1) = 1d0/Mmat(1,1)
else if (countcomp == 2) then
  a = Mmat(1,1)
  b = Mmat(1,2)
  c = Mmat(2,1)
  d = Mmat(2,2)
  determinant = (a*d) - (b*c)
  Mmat(1,1) = d/determinant
  Mmat(1,2) = -1d0*(b/determinant)
  Mmat(2,1) = -1d0*(c/determinant)
  Mmat(2,2) = a/determinant
else if (countcomp == 3) then
  Cofactor(1,1) = ((Mmat(2,2)*Mmat(3,3)) - (Mmat(2,3)*Mmat(3,2)))
  Cofactor(1,2) = -1d0*((Mmat(2,1)*Mmat(3,3)) - (Mmat(2,3)*Mmat(3,1)))
  Cofactor(1,3) = ((Mmat(2,1)*Mmat(3,2)) - (Mmat(2,2)*Mmat(3,1)))

  Cofactor(2,1) = -1d0*((Mmat(1,2)*Mmat(3,3)) - (Mmat(1,3)*Mmat(3,2)))
  Cofactor(2,2) = ((Mmat(1,1)*Mmat(3,3)) - (Mmat(1,3)*Mmat(3,1)))
  Cofactor(2,3) = -1d0*((Mmat(1,1)*Mmat(3,2)) - (Mmat(1,2)*Mmat(3,1)))

  Cofactor(3,1) = ((Mmat(1,2)*Mmat(2,3)) - (Mmat(1,3)*Mmat(2,2)))
  Cofactor(3,2) = -1d0*((Mmat(1,1)*Mmat(2,3)) - (Mmat(1,3)*Mmat(2,1)))
  Cofactor(3,3) = ((Mmat(1,1)*Mmat(2,2)) - (Mmat(1,2)*Mmat(2,1)))

  determinant = Mmat(1,1)*Cofactor(1,1) + Mmat(1,2)*Cofactor(1,2)
              + Mmat(1,3)*Cofactor(1,3)

  do i = 1,3
    do j = 1,3
      Mmat(i,j) = Cofactor(j,i)/determinant
    end do
  end do
end if
return
end
!=============================================================
subroutine GETPmatrix(seed,Pmat)
implicit none
integer :: i,j,k,a
integer, dimension(3) :: seed
double precision ::UNIFORM,sum,L,U
double precision, dimension(4,4,4) :: PMat

do k = 1,2
  do i = 1,4
    if(i == 1) then
    PMat(k,i,1) = 0.00000001d0
    else if(i > 1) then
```

```
      L = 0.00001
      U = 0.00002
      call GETuniform(seed,L,U,UNIFORM)
      PMat(k,i,1) = UNIFORM
    end if
    do j = 2,4
      L = 0.1
      U = 0.9
      call GETuniform(seed,L,U,UNIFORM)
      PMat(k,i,j) = UNIFORM
    end do
  end do
end do

do k = 1,2
  do i = 1,4
    sum = 0d0
    do j = 1,4
      sum = sum + PMat(k,i,j)
    end do
    do j = 1,4
      PMat(k,i,j) = Pmat(k,i,j)*(1.0d0/sum)
    end do
  end do
end do

L = 0.1
U = 0.9

  do k = 3,4
    do i = 1,4
      do j = 1,4
        call GETuniform(seed,L,U,UNIFORM)
        PMat(k,i,j) = UNIFORM
      end do
    end do
  end do

  do k = 3,4
    do i = 1,4
      sum = 0d0
      do j = 1,4
        sum = sum + PMat(k,i,j)
      end do
      do j = 1,4
        PMat(k,i,j) = Pmat(k,i,j)*(1.0d0/sum)
      end do
    end do
  end do
return
end
!================================================================
subroutine GETRmatrix(seed,RMat)
implicit none
integer :: i,j
integer, dimension(3) :: seed
double precision :: L,U,x
double precision, dimension(4,4) :: RMat

do i = 1,2
```

```
 if(i == 1) then
   L = 7.5d0
   U = 8.0d0
 else if(i == 2) then
   L = 6.5d0
   U = 7.0d0
 end if
 call GETuniform(seed,L,U,x)
 RMat(i,1) = x
end do

L = 0.00001
U = 0.00002

do i = 1,2
  do j = 2,4
    call GETuniform(seed,L,U,x)
    RMat(i,j) = x
  end do
end do

do i = 3,4
 if(i == 3) then
   L = 5.0d0
   U = 5.5d0
 else if(i == 4) then
   L = 3.0d0
   U = 3.5d0
 end if
   do j = 1,4
     call GETuniform(seed,L,U,x)
     RMat(i,j) = x
   end do
end do
return
end
!=============================================================
subroutine GETuniform(seed,L,U,x)
implicit none
integer, dimension(3) :: seed
double precision :: r,s,x,L,U

seed(1) = mod(171*seed(1),30269)
seed(2) = mod(172*seed(2),30307)
seed(3) = mod(170*seed(3),30323)

s = seed(1)*1d0/30269 + seed(2)*1d0/30307 + seed(3)*1d0/30323
r = s - int(s)

x = L + (U-L)*r
return
end
!=============================================================
subroutine GETrandom(seed,x)
implicit none
integer :: x
integer, dimension(3) :: seed
double precision :: r,s

seed(1) = mod(171*seed(1),30269)
```

```
seed(2) = mod(172*seed(2),30307)
seed(3) = mod(170*seed(3),30323)

s = seed(1)*1d0/30269 + seed(2)*1d0/30307 + seed(3)*1d0/30323
r = s - int(s)

x = int(16*r) return end
!=============================================================
subroutine Check(IND,num)
implicit none
integer :: i,j,k,c,d,num
double precision :: max
integer, dimension(4,4) :: ORDER1,ORDER2
double precision, dimension(4,4) :: IND

c = 0
d = 0
ORDER1 = 0
do k = 1,16
  max = -100000
  do i = 1,4
    do j = 1,4
      if (IND(i,j) > max) then
        max = IND(i,j)
        c = i
        d = j
      end if
    end do
  end do
  ORDER1(c,d) = k
  IND(c,d) = 0
end do

c = 0
d = 0
ORDER2 = 0
do k = 1,16
  max = -100000
  do i = 1,4
    do j = 1,4
      if (IND(i,j) > max) then
        max = IND(i,j)
        c = i
        d = j
      end if
    end do
  end do
  ORDER2(c,d) = k
  IND(c,d) = 0
end do

num = 0
do i = 1,4
  do j = 1,4
    if(ORDER1(i,j) /= ORDER2(i,j)) then
      num = num + 1
    end if
  end do
end do
return
```

```
end
!=============================================================
subroutine GETbandit1(IND,bandit1,cond)
implicit none
integer :: i,j,c,cond,bandit1
double precision :: max
double precision, dimension(7) :: Gsub
double precision, dimension(8) :: min
double precision, dimension(4,4) :: IND

min = 0d0
do i= 1,4
  min(i) = 1000000d0
  do j = 1,4
    if (IND(i,j) < min(i)) then
      min(i) = IND(i,j)
    end if
  end do
end do

do i = 1,3
  max = -10000
  do j = 1,4
    if (min(j) > max) then
      max = min(j)
      Gsub(i) = min(j)
      c = j
    end if
  end do
  if (i == 1) then
    bandit1 = c
  end if
  min(c) = -100000
end do

if (Gsub(1) /= Gsub(2) .and. Gsub(2) /= Gsub(3) .and. Gsub(1) /= Gsub(3)) then
  cond = 1
else
  cond = 2
end if return end
!=============================================================
subroutine GETrewards
(c,Rmat,Pmat,INDEX,alpha,speed1,speed2,bandit1,Rewoptimal,Rewpolicy,Rewgittins)
implicit none
integer :: a,b,e,f,h,i,j,k,l,m,n,q,speed1,speed2,bandit1,bandit2,bplace
integer :: b1,b2
integer,dimension(3) :: v,uv,vu,cnew
integer,dimension(3) :: stind,bind
integer,dimension(4) :: c,IS,u
integer,dimension(4) :: st1,st2
double precision :: alpha,Rnm,maxi,Rewoptimal,Rewpolicy
double precision :: R12,R21,Rewgittins
double precision, dimension(2) :: dis
double precision, dimension(4) :: d
double precision, dimension(6) :: max,rew
double precision, dimension(10) :: PSumind
double precision, dimension(8) :: Psum
double precision, dimension(4,2) :: BP1,POL1
double precision, dimension(4,4) :: Rmat
double precision, dimension(4,4) :: INDEX
```

```
double precision, dimension(4,4,4) :: Pmat
double precision, dimension(4,4,4,2) :: BP2,POL2
double precision, dimension(4,4,4,4,2) :: O0,BP
double precision, dimension(4,4,4,4,2) :: I0
double precision, dimension(4,4,4,4,4,2) :: I1,I2,I3,I4,I5
double precision, dimension(4,4,4,4,4,2) :: O1,O2,O3,O4,O5


d(1) = (1.0d0/alpha)*(1.0d0 - exp(-1.0d0*(alpha/(1.0d0*speed1))))
d(2) = (1.0d0/alpha)*(1.0d0 - exp(-1.0d0*(alpha/(1.0d0*speed2))))
d(3) = (1.0d0/alpha)*(1.0d0 - exp(-1.0d0*(alpha/(1.0d0*speed1*speed2))))
d(4) = exp(-1.0d0*(alpha/(1.0d0*speed1*speed2)))

dis(1) = exp(-1.0d0*(alpha/(1.0d0*speed1)))
dis(2) = exp(-1.0d0*(alpha/(1.0d0*speed2)))

O0=0d0
O1=0d0
O2=0d0
O3=0d0
O4=0d0
O5=0d0
BP1=0d0
BP2=0d0
BP=0d0
POL1 = 0d0
POL2 = 0d0

do q = 1,int((1.0d0*speed1*speed2)*(-1.0d0/alpha)
        *log((alpha*10e-8)/((5.0d0/d(1))+(5.0d0/d(2)))))+1
  if (mod(q,2) == 1) then
    e = 1
    f = 2
  else if (mod(q,2) == 0) then
    e = 2
    f = 1
  end if
  do i = 1,4
    do j = 1,4
      do k = 1,4
        do l = 1,4
          IS = (/i,j,k,l/)
          max(6) = -10000d0
          rew = 0d0
          do m = 1,4
            do a = 1,5
              max(a) = -10000d0
            end do
!--------------------------
            if (m == bandit1) then
              h = 0
              Psum(7) = 0d0
              do a = 1,4
                if (a /= bandit1) then
                  h = h + 1
                  uv(h) = IS(a)
                  vu(h) = a
                end if
Psum(7) = Psum(7) + Pmat(m,IS(m),a)*BP1(a,e)
              end do
```

```
BP1(IS(bandit1),f) = Rmat(bandit1,IS(bandit1)) + dis(1)*Psum(7)
              maxi = -10000d0
              bandit2 = 0
              bplace = 0
              do b = 1,3
                if (INDEX(vu(b),uv(b)) > maxi) then
                  maxi = INDEX(vu(b),uv(b))
                  bandit2 = vu(b)
                  bplace = b
                end if
              end do
            end if
            h = 0
!---------------------------
            do n = 1,4
              if (n /= m) then
!---------------------------
                h = h + 1
!---------------------------
Rnm = ((Rmat(n,IS(n))/d(1)) + (Rmat(m,IS(m))/d(2)))*d(3)
                do b = 1,8
                  PSum(b) = 0d0
                end do
                do a = 1,4
!---------------------------
                  if (m == bandit1 .and. n == bandit2) then
                    v = uv
                    v(bplace) = a
Psum(8) = Psum(8) + Pmat(bandit2,IS(bandit2),a)*BP2(v(1),v(2),v(3),e)
                  end if
!---------------------------
                  u = IS
                  u(n) = a
Psum(1) = Psum(1) + Pmat(n,IS(n),a)*01(u(1),u(2),u(3),u(4),m,e)
Psum(2) = Psum(2) + Pmat(n,IS(n),a)*02(u(1),u(2),u(3),u(4),m,e)
Psum(3) = Psum(3) + Pmat(n,IS(n),a)*03(u(1),u(2),u(3),u(4),m,e)
Psum(4) = Psum(4) + Pmat(n,IS(n),a)*04(u(1),u(2),u(3),u(4),m,e)
Psum(5) = Psum(5) + Pmat(n,IS(n),a)*05(u(1),u(2),u(3),u(4),m,e)
                  do b = 1,4
                    u(m) = b
Psum(6) = Psum(6) + Pmat(n,IS(n),a)*Pmat(m,IS(m),b)*00(u(1),u(2),u(3),u(4),e)
                  end do
                end do
rew(1) = Rnm + d(4)*PSum(2)
rew(2) = Rnm + d(4)*PSum(3)
rew(3) = Rnm + d(4)*PSum(4)
rew(4) = Rnm + d(4)*PSum(5)
rew(5) = Rnm + d(4)*PSum(6)
rew(6) = Rnm + d(4)*PSum(1)
!------------------------
                if (m == bandit1 .and. n == bandit2) then
BP2(uv(1),uv(2),uv(3),f) = Rmat(bandit2,IS(bandit2)) + dis(2)*Psum(8)
                end if
!------------------------
                do a = 1,6
                  if (rew(a) > max(a)) then
                    max(a) = rew(a)
                  end if
                end do
              end if
```

```
            end do
O1(i,j,k,l,m,f) = max(1)
O2(i,j,k,l,m,f) = max(2)
O3(i,j,k,l,m,f) = max(3)
O4(i,j,k,l,m,f) = max(4)
O5(i,j,k,l,m,f) = max(5)
          end do
O0(i,j,k,l,f) = max(6)
!-------------------------------------------------------------------------
!                          GITTINS INDICES BIT
!-------------------------------------------------------------------------
          maxi = -10000d0
          do a = 1,4
            if(INDEX(a,IS(a)) > maxi) then
              maxi = INDEX(a,IS(a))
              b1 = a
            end if
          end do
          h = 0
          do a = 1,4
            if(a /= b1) then
              h = h + 1
              stind(h) = IS(a)
              bind(h) = a
            end if
          end do
          maxi = -10000d0
          do a = 1,3
            if(INDEX(bind(a),stind(a)) > maxi ) then
              maxi = INDEX(bind(a),stind(a))
              b2 = bind(a)
            end if
          end do
R12 = Rmat(b1,IS(b1))*d(3)/d(1) + Rmat(b2,IS(b2))*d(3)/d(2)
R21 = Rmat(b2,IS(b2))*d(3)/d(1) + Rmat(b1,IS(b1))*d(3)/d(2)
          PSumind = 0d0
          do a = 1,4
            st1 = IS
            st1(b1) = a
PSumind(1) = PSumind(1) + Pmat(b1,IS(b1),a)*I1(st1(1),st1(2),st1(3),st1(4),b2,e)
PSumind(2) = PSumind(2) + Pmat(b1,IS(b1),a)*I2(st1(1),st1(2),st1(3),st1(4),b2,e)
PSumind(4) = PSumind(4) + Pmat(b1,IS(b1),a)*I3(st1(1),st1(2),st1(3),st1(4),b2,e)
PSumind(6) = PSumind(6) + Pmat(b1,IS(b1),a)*I4(st1(1),st1(2),st1(3),st1(4),b2,e)
PSumind(8) = PSumind(8) + Pmat(b1,IS(b1),a)*I5(st1(1),st1(2),st1(3),st1(4),b2,e)
            st2 = IS
            st2(b2) = a
PSumind(3) = PSumind(3) + Pmat(b2,IS(b2),a)*I2(st2(1),st2(2),st2(3),st2(4),b1,e)
PSumind(5) = PSumind(5) + Pmat(b2,IS(b2),a)*I3(st2(1),st2(2),st2(3),st2(4),b1,e)
PSumind(7) = PSumind(7) + Pmat(b2,IS(b2),a)*I4(st2(1),st2(2),st2(3),st2(4),b1,e)
PSumind(9) = PSumind(9) + Pmat(b2,IS(b2),a)*I5(st2(1),st2(2),st2(3),st2(4),b1,e)
            do b = 1,4
              st1(b2) = b
PSumind(10) = PSumind(10) + Pmat(b1,IS(b1),a)*Pmat(b2,IS(b2),b)
             *I0(st1(1),st1(2),st1(3),st1(4),e)
            end do
          end do

I0(i,j,k,l,f) = R12 + d(4)*PSumind(1)
I1(i,j,k,l,b2,f) = R12 + d(4)*PSumind(2)
I1(i,j,k,l,b1,f) = R21 + d(4)*PSumind(3)
```

```
I2(i,j,k,l,b2,f) = R12 + d(4)*PSumind(4)
I2(i,j,k,l,b1,f) = R21 + d(4)*PSumind(5)
I3(i,j,k,l,b2,f) = R12 + d(4)*PSumind(6)
I3(i,j,k,l,b1,f) = R21 + d(4)*PSumind(7)
I4(i,j,k,l,b2,f) = R12 + d(4)*PSumind(8)
I4(i,j,k,l,b1,f) = R21 + d(4)*PSumind(9)
I5(i,j,k,l,b2,f) = R12 + d(4)*PSumind(10)
I5(i,j,k,l,b1,f) = R21 + d(4)*PSumind(10)
!----------------------------------------
        end do
      end do
    end do
  end do

  if(q == int((speed2*1d0)*(-1.0d0/alpha)
        *log((alpha*10e-8)/(5.0d0/d(2))))+1) then
  POL2 = BP2
  end if
  if(q == int((speed1*1d0)*(-1.0d0/alpha)
        *log((alpha*10e-8)/(5.0d0/d(1))))+1) then
  POL1 = BP1
  end if

end do

Rewoptimal = OO(c(1),c(2),c(3),c(4),f)
Rewgittins = IO(c(1),c(2),c(3),c(4),f)

h = 0
do a = 1,4
  if (a /= bandit1) then
    h = h + 1
    cnew(h) = c(a)
    v(h) = a
  end if
end do

Rewpolicy = POL1(c(bandit1),f) + POL2(cnew(1),cnew(2),cnew(3),f)
return
end
!=========================================================================
```

Appendix F

```
program spd_opt_limit_vsGittins_5&2

implicit none integer ::
i,j,k,l,h,d,b,type,speed1,speed2,count,countcomp,E,M,SIZE
integer :: num,RANDOM,y,cond,c,bandit1,SIM
integer, dimension(3) :: seed
integer, dimension(4) :: argmax,STATE
double precision :: SMALL,BIG,MEAN,MEDIAN,STD,t,alpha,Z,sum,sum2,PERCENT
double precision :: SMALL2,BIG2,MEAN2,MEDIAN2,STD2,PERCENT2,t1,t2
double precision :: SMALL3,BIG3,MEAN3,MEDIAN3,STD3,PERCENT3,t3,t4
double precision :: Ropt,Rind,Reward1,Reward2,Reward3,Rgit
double precision, dimension(4) :: r,G,IND
double precision, dimension(4,4) :: P,Rmat
integer, dimension(15,4) :: S
double precision, dimension(15,4) :: A,AV
double precision, dimension(4,4) :: INDEX,INDEX2
double precision, dimension(4,4,4) :: Pmat
double precision, dimension(1000) :: Subopt,ORDER,Subopt2,ORDER2,Subopt3,ORDER3

type = 7
SIM = 1000
speed1 = 5
speed2 = 2
seed = (/21681,3814,14529/)
E = 4

do j = 15,1,-1
  d = j
  do i = 3,0,-1
    t = d/2**i
    if (t < 1) S(j,(4-i)) = 0
    if (t>= 1) then
      S(j,(4-i)) = 1
      d = mod(d,2**i)
    end if
  end do
end do

do c = 1,7
  if (c == 1) then
    alpha = 0.5d0
  else if (c == 2) then
    alpha = 0.25d0
  else if (c == 3) then
    alpha = 0.1d0
  else if (c == 4) then
    alpha = 0.05d0
  else if (c == 5) then
```

```
  alpha = 0.025d0
else if (c == 6) then
  alpha = 0.01d0
else if (c == 7) then
  alpha = 0.005d0
end if

Subopt = 0d0
Subopt2 = 0d0
Subopt3 = 0d0
order = 0d0
order2 = 0d0
order3 = 0d0
SMALL = 1.0e+20
BIG = -1*(1.0e+20)
SMALL2 = 1.0e+20
BIG2 = -1*(1.0e+20)
SMALL3 = 1.0e+20
BIG3 = -1*(1.0e+20)

do y = 1,SIM
  Ropt = 0d0
  Rind = 0d0
  Reward1 = 0d0
  Reward2 = 0d0

  call GETrandom(seed,RANDOM)

  h = 0
  do i = 1,4
    do j = 1,4
      h = h + 1
      if (h == RANDOM) then
        STATE = (/1,1,i,j/)
      end if
    end do
  end do

  call GETPmatrix(seed,Pmat)
  call GETRmatrix(seed,Rmat)

    do b = 1,4
      do i = 1,4
        do j = 1,4
          P(i,j) = Pmat(b,i,j)
        end do
        r(i) = Rmat(b,i)
      end do
      h = 15
      G = 0
      IND = 0
      argmax = 0
      A = 0
      A(15,:) = (/1,1,1,1/)
      AV = 0
      do k = 1,E
        if (k > 1) call GETsubset(E,argmax,k,h)
          count = 0
          do i = 1,E
            count = count + S(h,i)
```

```
            countcomp = E - count
          end do
          if (k > 1) call GETAmatrix_LIMIT(AV,S,count,countcomp,P,h,A)
          call GETindices(b,k,S,h,r,A,argmax,G,IND,INDEX)
        end do
      end do

    INDEX2 = INDEX

    call Getbandit1(INDEX2,bandit1,cond)
    call GETrewards
    (STATE,Rmat,Pmat,INDEX2,alpha,speed1,speed2,bandit1,Ropt,Rind,Rgit)

    Subopt(y) = Ropt - Rind
    Subopt2(y) = Ropt - Rgit
    Subopt3(y) = Rind - Rgit

    if(Subopt(y) >= BIG) then
      BIG = Subopt(y)
    end if
    if(Subopt(y) <= SMALL) then
      SMALL = Subopt(y)
    end if
    if(Subopt2(y) >= BIG2) then
      BIG2 = Subopt2(y)
    end if
    if(Subopt2(y) <= SMALL2) then
      SMALL2 = Subopt2(y)
    end if
    if(Subopt3(y) >= BIG3) then
      BIG3 = Subopt3(y)
    end if
    if(Subopt3(y) <= SMALL3) then
      SMALL3 = Subopt3(y)
    end if

end do

ORDER = 1.0e+20

do j = 1,SIM
  do i = 1,SIM
    if(ORDER(j) >= Subopt(i)) then
      ORDER(j) = Subopt(i)
      k = i
    end if
  end do
  Subopt(k) = 1.0e+21
end do

j = 0
do i = 1,SIM
  if(ORDER(i) > 10e-8) j = j + 1
end do
if (mod(j,2) == 1 .and. j > 0)then
  MEDIAN = ORDER(SIM - j + ((j+1)/2))
else if (mod(j,2) == 0 .and. j > 0)then
  MEDIAN = (ORDER(SIM - j + (j/2)) + ORDER(SIM - j + ((j/2)+1)))/2d0
else if (j == 0) then
  MEDIAN = 0d0
```

```
end if

sum = 0d0
sum2 = 0d0
do i = 1,SIM
  sum = sum + ORDER(i)
  sum2 = sum2 + (ORDER(i))**2
end do

MEAN = sum/SIM*1d0
t = (((1d0/SIM*1d0)*sum2)) - (MEAN**2)
if(t > 0)then
  STD = sqrt((((1d0/SIM*1d0)*sum2)) - (MEAN**2))
else
  STD = 0d0
end if
PERCENT = 100.0d0 -(100d0*j/SIM*1d0)

write(unit = 6, fmt= "(a,f6.4)")"alpha = " ,alpha
write(unit = 6, fmt= "(a)")"========================================="
write(unit = 6, fmt= "(a)")"STATISTICS FOR Roptimal - Rpolicy "
write(unit = 6, fmt= "(a,f12.6,a,f12.6,a,f12.6)")
"(1) Minimum", SMALL,"     (2) Median ",MEDIAN,"     (3) Maximum",BIG
write(unit = 6, fmt= "(a,f12.6,a,f12.6,a,f12.6)")
"(4) Mean   ", MEAN,"     (5) Std Dev",STD,"     (6) Percent",PERCENT
write(unit = 6, fmt= "(a)")"========================================="

open(unit = 7, file = "US_vs_GITTINS_5&2.dat")

write(unit = 7, fmt= "(a,f6.4)")" alpha = " ,alpha
write(unit = 7, fmt= "(a)")"========================================="
write(unit = 7, fmt= "(a)")"STATISTICS FOR Roptimal - Rpolicy "
write(unit = 7, fmt= "(a,f12.6,a,f12.6,a,f12.6)")
"(1) Minimum", SMALL,"     (2) Median ",MEDIAN,"    (3) Maximum",BIG
write(unit = 7, fmt= "(a,f12.6,a,f12.6,a,f12.6)")
"(4) Mean   ", MEAN,"     (5) Std Dev",STD,"    (6) Percent",PERCENT
write(unit = 7, fmt= "(a)")"========================================="

ORDER2 = 1.0e+20
do j = 1,SIM
  do i = 1,SIM
    if(ORDER2(j) >= Subopt2(i)) then
      ORDER2(j) = Subopt2(i)
     k = i
    end if
 end do
  Subopt2(k) = 1.0e+21
end do

j = 0
do i = 1,SIM
  if(ORDER2(i) > 10e-8) j = j + 1
end do
if (mod(j,2) == 1 .and. j > 0)then
  MEDIAN2 = ORDER2(SIM - j + ((j+1)/2))
else if (mod(j,2) == 0 .and. j > 0)then
  MEDIAN2 = (ORDER2(SIM - j + (j/2)) + ORDER2(SIM - j + ((j/2)+1)))/2d0
else if (j == 0) then
  MEDIAN2 = 0d0
end if
```

```
sum = 0d0
sum2 = 0d0
do i = 1,SIM
  sum = sum + ORDER2(i)
  sum2 = sum2 + ((ORDER2(i))**2)
end do

MEAN2 = (sum/SIM*1d0)
t2 = (((1d0/SIM*1d0)*sum2)) - (MEAN2**2)

if(t2 > 0)then
  STD2 = sqrt((((1d0/SIM*1d0)*sum2)) - ((MEAN2)**2))
else
  STD2 = 0d0
end if
PERCENT2 = 100.0d0 - (100.0d0*j/SIM*1.0d0)

write(unit = 6, fmt= "(a)")"STATISTICS FOR Roptimal - Rgittins "
write(unit = 6, fmt= "(a,f12.6,a,f12.6,a,f12.6)")
"(1) Minimum", SMALL2,"    (2) Median ",MEDIAN2,"     (3) Maximum",BIG2
write(unit = 6, fmt= "(a,f12.6,a,f12.6,a,f12.6)")
"(4) Mean   ", MEAN2,"    (5) Std Dev",STD2,"     (6) Percent",PERCENT2
write(unit = 6, fmt= "(a)")"========================================="

write(unit = 7, fmt= "(a)")"STATISTICS FOR Roptimal - Rgittins "
write(unit = 7, fmt= "(a,f12.6,a,f12.6,a,f12.6)")
"(1) Minimum", SMALL2,"    (2) Median ",MEDIAN2,"     (3) Maximum",BIG2
write(unit = 7, fmt= "(a,f12.6,a,f12.6,a,f12.6)")
"(4) Mean   ", MEAN2,"    (5) Std Dev",STD2,"     (6) Percent",PERCENT2
write(unit = 7, fmt= "(a)")"========================================="

ORDER3 = 1.0e+20
do j = 1,SIM
  do i = 1,SIM
    if(ORDER3(j) >= Subopt3(i)) then
      ORDER3(j) = Subopt3(i)
     k = i
    end if
 end do
  Subopt3(k) = 1.0e+21
end do

j = 0
do i = 1,SIM
  if(ORDER3(i) > 10e-8) j = j + 1
end do

MEDIAN3 = ((ORDER3(SIM/2)) + ORDER3((SIM/2)+1))/2d0

sum = 0d0
sum2 = 0d0
do i = 1,SIM
  sum = sum + ORDER3(i)
  sum2 = sum2 + ((ORDER3(i))**2)
end do

MEAN3 = (sum/SIM*1d0)
t3 = (((1d0/SIM*1d0)*sum2)) - (MEAN3**2)
```

```
  if(t3 > 0)then
    STD3 = sqrt((((1d0/SIM*1d0)*sum2)) - ((MEAN3)**2))
  else
    STD3 = 0d0
  end if
  PERCENT3 = 100.0d0 - (100.0d0*j/SIM*1.0d0)

  write(unit = 6, fmt= "(a)")"STATISTICS FOR Rpolicy - Rgittins "
  write(unit = 6, fmt= "(a,f12.6,a,f12.6,a,f12.6)")
  "(1) Minimum", SMALL3,"    (2) Median ",MEDIAN3,"     (3) Maximum",BIG3
  write(unit = 6, fmt= "(a,f12.6,a,f12.6,a,f12.6)")
  "(4) Mean   ", MEAN3,"    (5) Std Dev",STD3,"     (6) Percent",PERCENT3
  write(unit = 6, fmt= "(a)")"========================================="

  write(unit = type, fmt= "(a)")"STATISTICS FOR Rpolicy - Rgittins "
  write(unit = type, fmt= "(a,f12.6,a,f12.6,a,f12.6)")
  "(1) Minimum", SMALL3,"    (2) Median ",MEDIAN3,"     (3) Maximum",BIG3
  write(unit = type, fmt= "(a,f12.6,a,f12.6,a,f12.6)")
  "(4) Mean   ", MEAN3,"    (5) Std Dev",STD3,"     (6) Percent",PERCENT3
  write(unit = type, fmt= "(a)")"========================================="

end do

end program
!=====================================================================
subroutine GETsubset(E,argmax,k,h)
implicit none
integer :: k,h,E
integer, dimension(4) :: argmax

h = h - 2**(E - argmax(k - 1))
return
end
!=====================================================================
subroutine GETAmatrix_LIMIT(AV,S,count,countcomp,P,h,A)
implicit none
integer :: d,g,i,j,h,f,k,count,countcomp,E
double precision :: matsum,Z,sum
integer, dimension(15,4) :: S
double precision, dimension(4,4) :: P
double precision, dimension(15,4) :: A,AV
double precision, dimension(countcomp,countcomp) :: Mmat,Pmat
double precision, dimension(countcomp,1) :: Pvec,Psum,Product
double precision, dimension(count) :: V1,V2

E = 4
!-------------------------------------------------
Mmat = 0d0
!-------------------------------------------------
d = 0
do i = 1,E
  if(S(h,i) == 0) then
    if(d < countcomp) d = d + 1
    Mmat(d,d) = 1 - P(i,i)
    g = d
    do j = 1,E
      if (i < j) then
        if (S(h,j) == 0) then
          if (g < countcomp) g = g + 1
          Mmat(d,g) = (-1.0d0)*P(i,j)
```

```
          Mmat(g,d) = (-1.0d0)*P(j,i)
        end if
      end if
    end do
  end if
end do
!-------------------------------------------------
Psum = 0d0
Pvec = 0d0
Pmat = 0d0
d = 0
do i = 1,E
  g = 0
  if (S(h,i) == 0) then
    if (d < countcomp) d = d + 1
    do j = 1,E
      if (S(h,j) == 0) then
        if(g < countcomp) g = g + 1
        Pmat(d,g) = P(i,j)
      end if
    end do
  end if
end do
!-------------------------------------------------
do i = 1,countcomp
  do j = 1,countcomp
    Psum(i,1) = Psum(i,1) + Pmat(i,j)
  end do
end do
do i = 1,countcomp
  Pvec(i,1) = (1 - Psum(i,1))
end do
!-------------------------------------------------
Pvec = 1 + Pvec
!-------------------------------------------------
call GETinverse(countcomp,Mmat)
Product = 0d0
do i = 1,countcomp
  sum = 0d0
  do j = 1,countcomp
    sum = sum + (Mmat(i,j)*Pvec(j,1))
  end do
  Product(i,1) = sum
end do Pvec = Product
!-------------------------------------------------
f = 0
do k = 1,4
  if (S(h,k) == 0) then
    if (f < countcomp) f = f + 1
    AV(h,k) = Pvec(f,1)
  end if
end do

d = 0
do i = 1,E
  matsum = 0
  if (S(h,i) == 1) then
    if (d < count) d = d + 1
    do j = 1,E
      if (S(h,j) == 0) matsum = matsum + (P(i,j) * AV(h,j))
```

```
    end do
    V1(d) = matsum
    matsum = 0
    do j = 1,E
      if (S(h,j) == 1) matsum = matsum + (P(i,j))
    end do
    V2(d) = matsum
    AV(h,i) = V1(d) + V2(d)
  end if
end do

AV(15,:) = (/1,1,1,1/)

f = 0
do k = 1,4
  if (S(h,k) == 1) then
    if (f < count) f = f + 1
    A(h,k) = AV(h,k)
  end if
end do
return
end
!=====================================================================
subroutine GETindices(b,k,S,h,r,A,argmax,G,IND,INDEX)
implicit none
integer :: b,i,h,k,l,j,E
double precision :: max,sum
integer, dimension(4) :: argmax
double precision, dimension(4) :: r,G,IND
double precision, dimension(15,4) :: A integer,
dimension(15,4) :: S
double precision, dimension(4,4) :: INDEX

if (k == 1) then
  max = -1000000
  do i = 1,4
    if (r(i)/A(h,i) > max) then
      max = r(i)/A(h,i)
      argmax(1) = i
    end if
  end do
  G(argmax(1)) = max
  IND(argmax(1)) = G(argmax(1))
else
  E = 4
  max = -1000000
  do i = 1,4
    l = 15
    sum = 0
    if (S(h,i) == 1) then
      do j = 1,k-1
        sum = sum + A(l,i)*G(argmax(j))
        l = l - 2**(E - argmax(j))
      end do
      if ((r(i) - sum)/A(h,i) > max) then
        max = (r(i) - sum)/A(h,i)
        argmax(k) = i
      end if
    end if
  end do
```

```
  G(argmax(k)) = max
  IND(argmax(k)) = G(argmax(k)) + IND(argmax(k - 1))
end if

INDEX(b,argmax(k)) = IND(argmax(k))
return
end
!=============================================================
subroutine GETinverse(countcomp,Mmat)
implicit none
integer :: i,j,countcomp
double precision :: a,b,c,d,determinant
double precision, dimension(countcomp,countcomp) :: Mmat
double precision, dimension(3,3) :: Cofactor

if (countcomp == 1) then
  Mmat(1,1) = 1d0/Mmat(1,1)
else if (countcomp == 2) then
  a = Mmat(1,1)
  b = Mmat(1,2)
  c = Mmat(2,1)
  d = Mmat(2,2)
  determinant = (a*d) - (b*c)
  Mmat(1,1) = d/determinant
  Mmat(1,2) = -1d0*(b/determinant)
  Mmat(2,1) = -1d0*(c/determinant)
  Mmat(2,2) = a/determinant
else if (countcomp == 3) then
  Cofactor(1,1) = ((Mmat(2,2)*Mmat(3,3)) - (Mmat(2,3)*Mmat(3,2)))
  Cofactor(1,2) = -1d0*((Mmat(2,1)*Mmat(3,3)) - (Mmat(2,3)*Mmat(3,1)))
  Cofactor(1,3) = ((Mmat(2,1)*Mmat(3,2)) - (Mmat(2,2)*Mmat(3,1)))

  Cofactor(2,1) = -1d0*((Mmat(1,2)*Mmat(3,3)) - (Mmat(1,3)*Mmat(3,2)))
  Cofactor(2,2) = ((Mmat(1,1)*Mmat(3,3)) - (Mmat(1,3)*Mmat(3,1)))
  Cofactor(2,3) = -1d0*((Mmat(1,1)*Mmat(3,2)) - (Mmat(1,2)*Mmat(3,1)))

  Cofactor(3,1) = ((Mmat(1,2)*Mmat(2,3)) - (Mmat(1,3)*Mmat(2,2)))
  Cofactor(3,2) = -1d0*((Mmat(1,1)*Mmat(2,3)) - (Mmat(1,3)*Mmat(2,1)))
  Cofactor(3,3) = ((Mmat(1,1)*Mmat(2,2)) - (Mmat(1,2)*Mmat(2,1)))

  determinant = Mmat(1,1)*Cofactor(1,1) + Mmat(1,2)*Cofactor(1,2)
              + Mmat(1,3)*Cofactor(1,3)

  do i = 1,3
    do j = 1,3
      Mmat(i,j) = Cofactor(j,i)/determinant
    end do
  end do
end if
return
end
!=============================================================
subroutine GETPmatrix(seed,Pmat)
implicit none
integer :: i,j,k,a
integer, dimension(3) :: seed
double precision ::UNIFORM,sum,L,U
double precision, dimension(4,4,4) :: PMat

do k = 1,2
```

```
  do i = 1,4
    if(i == 1) then
    PMat(k,i,1) = 0.0d0
    else if(i > 1) then
      L = 0.00001
      U = 0.00002
      call GETuniform(seed,L,U,UNIFORM)
      PMat(k,i,1) = UNIFORM
    end if
    do j = 2,4
      L = 0.1
      U = 0.9
      call GETuniform(seed,L,U,UNIFORM)
      PMat(k,i,j) = UNIFORM
    end do
  end do
end do

do k = 1,2
  do i = 1,4
    sum = 0d0
    do j = 1,4
      sum = sum + PMat(k,i,j)
    end do
    do j = 1,4
      PMat(k,i,j) = Pmat(k,i,j)*(1.0d0/sum)
    end do
  end do
end do

L = 0.1
U = 0.9

  do k = 3,4
    do i = 1,4
      do j = 1,4
        call GETuniform(seed,L,U,UNIFORM)
        PMat(k,i,j) = UNIFORM
      end do
    end do
  end do

  do k = 3,4
    do i = 1,4
      sum = 0d0
      do j = 1,4
        sum = sum + PMat(k,i,j)
      end do
      do j = 1,4
        PMat(k,i,j) = Pmat(k,i,j)*(1.0d0/sum)
      end do
    end do
  end do
return
end
!=============================================================
subroutine GETRmatrix(seed,RMat)
implicit none
integer :: i,j
integer, dimension(3) :: seed
```

```
double precision :: L,U,x
double precision, dimension(4,4) :: RMat

do i = 1,2
 if(i == 1) then
   L = 7.5d0
   U = 8.0d0
 else if(i == 2) then
   L = 6.5d0
   U = 7.0d0
 end if
 call GETuniform(seed,L,U,x)
 RMat(i,1) = x
end do

L = 0.00001
U = 0.00002
do i = 1,2
  do j = 2,4
    call GETuniform(seed,L,U,x)
    RMat(i,j) = x
  end do
end do

do i = 3,4
 if(i == 3) then
   L = 5.0d0
   U = 5.5d0
 else if(i == 4) then
   L = 4.0d0
   U = 4.5d0
 end if
   do j = 1,4
     call GETuniform(seed,L,U,x)
     RMat(i,j) = x
   end do
end do
return
end
!=============================================================
subroutine GETuniform(seed,L,U,x)
implicit none
integer, dimension(3) :: seed
double precision :: r,s,x,L,U

seed(1) = mod(171*seed(1),30269)
seed(2) = mod(172*seed(2),30307)
seed(3) = mod(170*seed(3),30323)

s = seed(1)*1d0/30269 + seed(2)*1d0/30307 + seed(3)*1d0/30323
r = s - int(s)

x = L + (U-L)*r
return
end
!=============================================================
subroutine GETrandom(seed,x)
implicit none
integer :: x
integer, dimension(3) :: seed
```

```
double precision :: r,s

seed(1) = mod(171*seed(1),30269)
seed(2) = mod(172*seed(2),30307)
seed(3) = mod(170*seed(3),30323)

s = seed(1)*1d0/30269 + seed(2)*1d0/30307 + seed(3)*1d0/30323
r = s - int(s)

x = int(16*r)
return
end
!=============================================================
subroutine Check(IND,num)
implicit none
integer :: i,j,k,c,d,num
double precision :: max
integer, dimension(4,4) :: ORDER1,ORDER2
double precision, dimension(4,4) :: IND

c = 0
d = 0
ORDER1 = 0
do k = 1,16
  max = -100000
  do i = 1,4
    do j = 1,4
      if (IND(i,j) > max) then
        max = IND(i,j)
        c = i
        d = j
      end if
    end do
  end do
  ORDER1(c,d) = k
  IND(c,d) = 0
end do

c = 0
d = 0
ORDER2 = 0
do k = 1,16
  max = -100000
  do i = 1,4
    do j = 1,4
      if (IND(i,j) > max) then
        max = IND(i,j)
        c = i
        d = j
      end if
    end do
  end do
  ORDER2(c,d) = k
  IND(c,d) = 0
end do

num = 0
do i = 1,4
  do j = 1,4
    if(ORDER1(i,j) /= ORDER2(i,j)) then
```

```
      num = num + 1
    end if
  end do
end do
return
end
!=================================================================
subroutine GETbandit1(IND,bandit1,cond)
implicit none
integer :: i,j,c,cond,bandit1
double precision :: max
double precision, dimension(3) :: Gsub
double precision, dimension(4) :: min
double precision, dimension(4,4) :: IND

min = 0d0
do i= 1,4
  min(i) = 1000000d0
  do j = 1,4
    if (IND(i,j) < min(i)) then
      min(i) = IND(i,j)
    end if
  end do
end do

do i = 1,3
  max = -10000
  do j = 1,4
    if (min(j) > max) then
      max = min(j)
      Gsub(i) = min(j)
      c = j
    end if
  end do
  if (i == 1) then
    bandit1 = c
  end if
  min(c) = -100000
end do

if (Gsub(1) /= Gsub(2) .and. Gsub(2) /= Gsub(3) .and. Gsub(1) /= Gsub(3)) then
  cond = 1
else
  cond = 2
end if
return
end
!=============================================================
subroutine GETrewards
(c,Rmat,Pmat,INDEX,alpha,speed1,speed2,bandit1,Rewoptimal,Rewpolicy,Rewgittins)
implicit none
integer :: a,b,e,f,h,i,j,k,l,m,n,q,speed1,speed2,bandit1,bandit2,bplace
integer :: b1,b2
integer,dimension(3) :: v,uv,vu,cnew
integer,dimension(3) :: stind,bind
integer,dimension(4) :: st1,st2
integer,dimension(4) :: c,IS,u
double precision :: alpha,Rmn,Rnm,maxi,Rewoptimal,Rewpolicy
double precision :: R12,R21,Rewgittins
double precision, dimension(2) :: dis
```

```
double precision, dimension(4) :: d
double precision, dimension(6) :: max,rew
double precision, dimension(8) :: Psum
double precision, dimension(10) :: PSumind
double precision, dimension(4,2) :: IP1,POL1
double precision, dimension(4,4) :: Rmat
double precision, dimension(4,4) :: INDEX
double precision, dimension(4,4,4) :: Pmat
double precision, dimension(4,4,4,2) :: IP2,POL2
double precision, dimension(4,4,4,4,2) :: O0,IP,I0,I1
double precision, dimension(4,4,4,4,4,2) :: O2,O4,O5,O6,O8,I2,I4,I5,I6,I8
double precision, dimension(4,4,4,4,4,4,2) :: O1,O3,O7,O9,I3,I7,I9

d(1) = (1.0d0/alpha)*(1.0d0 - exp(-1.0d0*(alpha/(1.0d0*speed1))))
d(2) = (1.0d0/alpha)*(1.0d0 - exp(-1.0d0*(alpha/(1.0d0*speed2))))
d(3) = (1.0d0/alpha)*(1.0d0 - exp(-1.0d0*(alpha/(1.0d0*speed1*speed2))))
d(4) = exp(-1.0d0*(alpha/(1.0d0*speed1*speed2)))

dis(1) = exp(-1.0d0*(alpha/(1.0d0*speed1)))
dis(2) = exp(-1.0d0*(alpha/(1.0d0*speed2)))

O0=0d0
O1=0d0
O2=0d0
O3=0d0
O4=0d0
O5=0d0
O6=0d0
O7=0d0
O8=0d0
O9=0d0
IP1=0d0
IP2=0d0
IP=0d0
POL1 = 0d0
POL2 = 0d0

do q = 1,int((1.0d0*speed1*speed2)*(-1.0d0/alpha)
        *log((alpha*10e-8)/((5.0d0/d(1))+(5.0d0/d(2)))))+1
  if (mod(q,2) == 1) then
    e = 1
    f = 2
  else if (mod(q,2) == 0) then
    e = 2
    f = 1
  end if
  do i = 1,4
    do j = 1,4
      do k = 1,4
        do l = 1,4
          IS = (/i,j,k,l/)
          max(1) = -10000d0
          rew = 0d0
          do m = 1,4
            do a = 2,6
              max(a) = -10000d0
            end do
!--------------------------
            if (m == bandit1) then
              h = 0
```

```
              Psum(7) = 0d0
              do a = 1,4
                if (a /= bandit1) then
                  h = h + 1
                  uv(h) = IS(a)
                  vu(h) = a
                end if
                Psum(7) = Psum(7) + Pmat(m,IS(m),a)*IP1(a,e)
              end do
IP1(IS(bandit1),f) = Rmat(bandit1,IS(bandit1)) + dis(1)*Psum(7)
              maxi = -10000d0
              bandit2 = 0
              bplace = 0
              do b = 1,3
                if (INDEX(vu(b),uv(b)) > maxi) then
                  maxi = INDEX(vu(b),uv(b))
                  bandit2 = vu(b)
                  bplace = b
                end if
              end do
            end if
            h = 0
!----------------------------
            do n = 1,4
              if (m /= n) then
!----------------------------
                h = h + 1
!----------------------------
Rmn = ((d(3)*Rmat(m,IS(m))/d(1)) + (d(3)*Rmat(n,IS(n))/d(2)))
Rnm = ((Rmat(n,IS(n))/d(1)) + (Rmat(m,IS(m))/d(2)))*d(3)
              do b = 1,8
                PSum(b) = 0d0
              end do
              do a = 1,4
!----------------------------
                if (m == bandit1 .and. n == bandit2) then
                  v = uv
                  v(bplace) = a
Psum(8) = Psum(8) + Pmat(bandit2,IS(bandit2),a)*IP2(v(1),v(2),v(3),e)
                end if
!----------------------------
                u = IS
                u(m) = a
Psum(1) = Psum(1) + Pmat(m,IS(m),a)*02(u(1),u(2),u(3),u(4),n,e)
Psum(2) = Psum(2) + Pmat(m,IS(m),a)*04(u(1),u(2),u(3),u(4),n,e)
Psum(3) = Psum(3) + Pmat(m,IS(m),a)*05(u(1),u(2),u(3),u(4),n,e)
Psum(4) = Psum(4) + Pmat(m,IS(m),a)*06(u(1),u(2),u(3),u(4),n,e)
Psum(5) = Psum(5) + Pmat(m,IS(m),a)*08(u(1),u(2),u(3),u(4),n,e)
                do b = 1,4
                  u(n) = b
Psum(6) = Psum(6) + Pmat(n,IS(n),b)*Pmat(m,IS(m),a)*00(u(1),u(2),u(3),u(4),e)
                end do
              end do
rew(1) = Rmn + d(4)*01(i,j,k,l,m,n,e)
rew(2) = Rnm + d(4)*03(i,j,k,l,n,m,e)
rew(3) = Rnm + d(4)*Psum(3)
rew(4) = Rmn + d(4)*Psum(4)
rew(5) = Rnm + d(4)*07(i,j,k,l,n,m,e)
rew(6) = Rnm + d(4)*09(i,j,k,l,n,m,e)
!-----------------------
```

```
                if (m == bandit1 .and. n == bandit2) then
IP2(uv(1),uv(2),uv(3),f) = Rmat(bandit2,IS(bandit2)) + dis(2)*Psum(8)
                end if
!------------------------
                do a = 1,6
                  if (rew(a) > max(a)) then
                    max(a) = rew(a)
                  end if
                end do
O1(i,j,k,l,m,n,f) = Rmn + d(4)*Psum(1)
O3(i,j,k,l,m,n,f) = Rmn + d(4)*Psum(2)
O7(i,j,k,l,m,n,f) = Rmn + d(4)*Psum(5)
O9(i,j,k,l,m,n,f) = Rmn + d(4)*Psum(6)
              end if
            end do
O2(i,j,k,l,m,f) = max(2)
O4(i,j,k,l,m,f) = max(3)
O5(i,j,k,l,m,f) = max(4)
O6(i,j,k,l,m,f) = max(5)
O8(i,j,k,l,m,f) = max(6)
          end do
O0(i,j,k,l,f) = max(1)
!-------------------------------------------------------------------------
!                          GITTINS INDICES BIT
!-------------------------------------------------------------------------
          maxi = -10000d0
          do a = 1,4
            if(INDEX(a,IS(a)) > maxi) then
              maxi = INDEX(a,IS(a))
              b1 = a
            end if
          end do
          h = 0
          do a = 1,4
            if(a /= b1) then
              h = h + 1
              stind(h) = IS(a)
              bind(h) = a
            end if
          end do
          maxi = -10000d0
          do a = 1,3
            if(INDEX(bind(a),stind(a)) > maxi ) then
              maxi = INDEX(bind(a),stind(a))
              b2 = bind(a)
            end if
          end do
R12 = Rmat(b1,IS(b1))*d(3)/d(1) + Rmat(b2,IS(b2))*d(3)/d(2)
R21 = Rmat(b2,IS(b2))*d(3)/d(1) + Rmat(b1,IS(b1))*d(3)/d(2)
          PSumind = 0d0
          do a = 1,4
            st1 = IS
            st1(b1) = a
PSumind(1) = PSumind(1) + Pmat(b1,IS(b1),a)*I2(st1(1),st1(2),st1(3),st1(4),b2,e)
PSumind(2) = PSumind(2) + Pmat(b1,IS(b1),a)*I4(st1(1),st1(2),st1(3),st1(4),b2,e)
PSumind(4) = PSumind(4) + Pmat(b1,IS(b1),a)*I5(st1(1),st1(2),st1(3),st1(4),b2,e)
PSumind(6) = PSumind(6) + Pmat(b1,IS(b1),a)*I6(st1(1),st1(2),st1(3),st1(4),b2,e)
PSumind(8) = PSumind(8) + Pmat(b1,IS(b1),a)*I8(st1(1),st1(2),st1(3),st1(4),b2,e)
            st2 = IS
            st2(b2) = a
```

```
PSumind(3) = PSumind(3) + Pmat(b2,IS(b2),a)*I4(st2(1),st2(2),st2(3),st2(4),b1,e)
PSumind(5) = PSumind(5) + Pmat(b2,IS(b2),a)*I5(st2(1),st2(2),st2(3),st2(4),b1,e)
PSumind(7) = PSumind(7) + Pmat(b2,IS(b2),a)*I6(st2(1),st2(2),st2(3),st2(4),b1,e)
PSumind(9) = PSumind(9) + Pmat(b2,IS(b2),a)*I8(st2(1),st2(2),st2(3),st2(4),b1,e)
            do b = 1,4
              st1(b2) = b
PSumind(10) = PSumind(10) + Pmat(b1,IS(b1),a)*Pmat(b2,IS(b2),b)
              *I0(st1(1),st1(2),st1(3),st1(4),e)
            end do
          end do

I0(i,j,k,l,f) = R12 + d(4)*I1(i,j,k,l,e)
I1(i,j,k,l,f) = R12 + d(4)*PSumind(1)
I2(i,j,k,l,b1,f) = R21 + d(4)*I3(i,j,k,l,b2,b1,e)
I2(i,j,k,l,b2,f) = R12 + d(4)*I3(i,j,k,l,b1,b2,e)
I3(i,j,k,l,b1,b2,f) = R12 + d(4)*PSumind(2)
I3(i,j,k,l,b2,b1,f) = R21 + d(4)*PSumind(3)
I4(i,j,k,l,b1,f) = R21 + d(4)*PSumind(4)
I4(i,j,k,l,b2,f) = R12 + d(4)*PSumind(5)
I5(i,j,k,l,b1,f) = R12 + d(4)*PSumind(6)
I5(i,j,k,l,b2,f) = R21 + d(4)*PSumind(7)
I6(i,j,k,l,b1,f) = R21 + d(4)*I7(i,j,k,l,b2,b1,f)
I6(i,j,k,l,b2,f) = R12 + d(4)*I7(i,j,k,l,b1,b2,f)
I7(i,j,k,l,b1,b2,f) = R12 + d(4)*PSumind(8)
I7(i,j,k,l,b2,b1,f) = R21 + d(4)*PSumind(9)
I8(i,j,k,l,b1,f) = R21 + d(4)*I9(i,j,k,l,b2,b1,e)
I8(i,j,k,l,b2,f) = R12 + d(4)*I9(i,j,k,l,b1,b2,e)
I9(i,j,k,l,b1,b2,f) = R12 + d(4)*PSumind(10)
I9(i,j,k,l,b2,b1,f) = R21 + d(4)*PSumind(10)
        end do
      end do
    end do
  end do

  if(q == int((speed2*1d0)*(-1.0d0/alpha)
        *log((alpha*10e-8)/(5.0d0/d(2))))+1) then
  POL2 = IP2
  end if
  if(q == int((speed1*1d0)*(-1.0d0/alpha)
        *log((alpha*10e-8)/(5.0d0/d(1))))+1) then
  POL1 = IP1
  end if

end do
Rewoptimal = O0(c(1),c(2),c(3),c(4),f)
Rewgittins = I0(c(1),c(2),c(3),c(4),f)

h = 0
do a = 1,4
  if (a /= bandit1) then
    h = h + 1
    cnew(h) = c(a)
    v(h) = a
  end if
end do Rewpolicy = POL1(c(bandit1),f) +
POL2(cnew(1),cnew(2),cnew(3),f)
return
end
!=======================================================================
```

Appendix G

```
program spd_opt_limit_vsGittins_4&3

implicit none
integer :: i,j,k,l,h,d,b,type,speed1,speed2,count,countcomp,E,M,SIZE
integer :: num,RANDOM,y,cond,c,bandit1,SIM
integer, dimension(3) :: seed
integer, dimension(4) :: argmax,STATE
double precision :: SMALL,BIG,MEAN,MEDIAN,STD,t,alpha,Z,sum,sum2,PERCENT
double precision :: SMALL2,BIG2,MEAN2,MEDIAN2,STD2,PERCENT2,t1,t2
double precision :: SMALL3,BIG3,MEAN3,MEDIAN3,STD3,PERCENT3,t3,t4
double precision :: Ropt,Rind,Reward1,Reward2,Reward3,Rgit
double precision, dimension(4) :: r,G,IND double precision,
dimension(4,4) :: P,Rmat
integer, dimension(15,4) :: S
double precision, dimension(15,4) :: A,AV
double precision, dimension(4,4) :: INDEX,INDEX2
double precision, dimension(4,4,4) :: Pmat
double precision, dimension(1000) :: Subopt,ORDER,Subopt2,ORDER2,Subopt3,ORDER3

type = 7
SIM = 1000
speed1 = 4
speed2 = 3
seed = (/6852,17564,27846/)
E = 4

do j = 15,1,-1
  d = j
  do i = 3,0,-1
    t = d/2**i
    if (t < 1) S(j,(4-i)) = 0
    if (t>= 1) then
      S(j,(4-i)) = 1
      d = mod(d,2**i)
    end if
  end do
end do

do c = 1,7
  if (c == 1) then
    alpha = 0.5d0
  else if (c == 2) then
    alpha = 0.25d0
  else if (c == 3) then
    alpha = 0.1d0
  else if (c == 4) then
    alpha = 0.05d0
  else if (c == 5) then
```

```
  alpha = 0.025d0
else if (c == 6) then
  alpha = 0.01d0
else if (c == 7) then
  alpha = 0.005d0
end if

Subopt = 0d0
Subopt2 = 0d0
Subopt3 = 0d0
order = 0d0
order2 = 0d0
order3 = 0d0
SMALL = 1.0e+20
BIG = -1*(1.0e+20)
SMALL2 = 1.0e+20
BIG2 = -1*(1.0e+20)
SMALL3 = 1.0e+20
BIG3 = -1*(1.0e+20)

do y = 1,SIM

  Ropt = 0d0
  Rind = 0d0
  Rgit = 0d0
  Reward1 = 0d0
  Reward2 = 0d0

  call GETrandom(seed,RANDOM)

  h = 0
  do i = 1,4
    do j = 1,4
      h = h + 1
      if (h == RANDOM) then
        STATE = (/1,1,i,j/)
      end if
    end do
  end do

  call GETPmatrix(seed,Pmat)
  call GETRmatrix(seed,Rmat)

    do b = 1,4
      do i = 1,4
        do j = 1,4
          P(i,j) = Pmat(b,i,j)
        end do
        r(i) = Rmat(b,i)
      end do
      h = 15
      G = 0
      IND = 0
      argmax = 0
      A = 0
      A(15,:) = (/1,1,1,1/)
      AV = 0
      do k = 1,E
        if (k > 1) call GETsubset(E,argmax,k,h)
          count = 0
```

```
          do i = 1,E
            count = count + S(h,i)
            countcomp = E - count
          end do
          if (k > 1) call GETAmatrix_LIMIT(AV,S,count,countcomp,P,h,A)
          call GETindices(b,k,S,h,r,A,argmax,G,IND,INDEX)
        end do
      end do

    INDEX2 = INDEX

    call Getbandit1(INDEX2,bandit1,cond)
    call GETrewards
    (STATE,Rmat,Pmat,INDEX2,alpha,speed1,speed2,bandit1,Ropt,Rind,Rgit)
    Subopt(y) = Ropt - Rind
    Subopt2(y) = Ropt - Rgit
    Subopt3(y) = Rind - Rgit

    if(Subopt(y) >= BIG) then
      BIG = Subopt(y)
    end if
    if(Subopt(y) <= SMALL) then
      SMALL = Subopt(y)
    end if
    if(Subopt2(y) >= BIG2) then
      BIG2 = Subopt2(y)
    end if
    if(Subopt2(y) <= SMALL2) then
      SMALL2 = Subopt2(y)
    end if
    if(Subopt3(y) >= BIG3) then
      BIG3 = Subopt3(y)
    end if
    if(Subopt3(y) <= SMALL3) then
      SMALL3 = Subopt3(y)
    end if

end do

ORDER = 1.0e+20

do j = 1,SIM
  do i = 1,SIM
    if(ORDER(j) >= Subopt(i)) then
      ORDER(j) = Subopt(i)
      k = i
    end if
  end do
  Subopt(k) = 1.0e+21
end do

j = 0
do i = 1,SIM
  if(ORDER(i) > 10e-8) j = j + 1
end do
if (mod(j,2) == 1 .and. j > 0)then
  MEDIAN = ORDER(SIM - j + ((j+1)/2))
else if (mod(j,2) == 0 .and. j > 0)then
  MEDIAN = (ORDER(SIM - j + (j/2)) + ORDER(SIM - j + ((j/2)+1)))/2d0
else if (j == 0) then
```

```
  MEDIAN = 0d0
end if

sum = 0d0
sum2 = 0d0
do i = 1,SIM
  sum = sum + ORDER(i)
  sum2 = sum2 + (ORDER(i))**2
end do

MEAN = sum/SIM*1d0
t = (((1d0/SIM*1d0)*sum2)) - (MEAN**2)
if(t > 0)then
  STD = sqrt((((1d0/SIM*1d0)*sum2)) - (MEAN**2))
else
  STD = 0d0
end if
PERCENT = 100.0d0 -(100d0*j/SIM*1d0)

write(unit = 6, fmt= "(a)")"--------------------------------------"
write(unit = 6, fmt= "(a)")"STATISTICS FOR Roptimal - Rpolicy "
write(unit = 6, fmt= "(a)")"--------------------------------------"
write(unit = 6, fmt= "(a,i2,a,i2,a,f6.4)")
"speed1 = ",speed1," speed2 = " ,speed2," alpha = " ,alpha
write(unit = 6, fmt= "(a)")"--------------------------------------"
write(unit = 6, fmt= "(a,i3)")" THE NUMBER OF SIMULATIONS IS  ", SIM
write(unit = 6, fmt= "(a,f12.6,a,f12.6,a,f12.6)")
"(1) Minimum", SMALL,"     (2) Median ",MEDIAN,"     (3) Maximum",BIG
write(unit = 6, fmt= "(a,f12.6,a,f12.6,a,f12.6)")
"(4) Mean   ", MEAN,"     (5) Std Dev",STD,"     (6) Percent",PERCENT
write(unit = 6, fmt= "(a)")"======================================="

open(unit = 7, file = "US_vs_GITTINS_4&3.dat")

write(unit = 7, fmt= "(a)")"--------------------------------------"
write(unit = 7, fmt= "(a)")"STATISTICS FOR Roptimal - Rpolicy "
write(unit = 7, fmt= "(a)")"--------------------------------------"
write(unit = 7e, fmt= "(a,i2,a,i2,a,f6.4)")
"speed1 = ",speed1," speed2 = " ,speed2," alpha = " ,alpha
write(unit = 7e, fmt= "(a)")"--------------------------------------"
write(unit = 7, fmt= "(a,i3)")" THE NUMBER OF SIMULATIONS IS  ",SIM
write(unit = 7, fmt= "(a,f12.6,a,f12.6,a,f12.6)")
"(1) Minimum", SMALL,"     (2) Median ",MEDIAN,"    (3) Maximum",BIG
write(unit = 7, fmt= "(a,f12.6,a,f12.6,a,f12.6)")
"(4) Mean   ", MEAN,"     (5) Std Dev",STD,"    (6) Percent",PERCENT
write(unit = 7, fmt= "(a)")"======================================="
!-----------------------------------------------------------------------
ORDER2 = 1.0e+20
do j = 1,SIM
  do i = 1,SIM
    if(ORDER2(j) >= Subopt2(i)) then
      ORDER2(j) = Subopt2(i)
     k = i
    end if
 end do
  Subopt2(k) = 1.0e+21
end do

j = 0
do i = 1,SIM
```

```
    if(ORDER2(i) > 10e-8) j = j + 1
  end do
 if (mod(j,2) == 1 .and. j > 0)then
   MEDIAN2 = ORDER2(SIM - j + ((j+1)/2))
  else if (mod(j,2) == 0 .and. j > 0)then
    MEDIAN2 = (ORDER2(SIM - j + (j/2)) + ORDER2(SIM - j + ((j/2)+1)))/2d0
  else if (j == 0) then
    MEDIAN2 = 0d0
  end if

  sum = 0d0
  sum2 = 0d0
  do i = 1,SIM
    sum = sum + ORDER2(i)
    sum2 = sum2 + ((ORDER2(i))**2)
  end do

  MEAN2 = (sum/SIM*1d0)
  t2 = (((1d0/SIM*1d0)*sum2)) - (MEAN2**2)

  if(t2 > 0)then
    STD2 = sqrt((((1d0/SIM*1d0)*sum2)) - ((MEAN2)**2))
  else
    STD2 = 0d0
  end if
  PERCENT2 = 100.0d0 - (100.0d0*j/SIM*1.0d0)

  write(unit = 6, fmt= "(a)")"----------------------------------------"
  write(unit = 6, fmt= "(a)")"STATISTICS FOR Roptimal - Rgittins "
  write(unit = 6, fmt= "(a)")"----------------------------------------"
  write(unit = 6, fmt= "(a,i2,a,i2,a,f6.4)")
  "speed1 = ",speed1," speed2 = " ,speed2," alpha = " ,alpha
  write(unit = 6, fmt= "(a)")"----------------------------------------"
  write(unit = 6, fmt= "(a,i3)")" THE NUMBER OF SIMULATIONS IS  ", SIM
  write(unit = 6, fmt= "(a,f12.6,a,f12.6,a,f12.6)")
  "(1) Minimum", SMALL2,"    (2) Median ",MEDIAN2,"     (3) Maximum",BIG2
  write(unit = 6, fmt= "(a,f12.6,a,f12.6,a,f12.6)")
  "(4) Mean   ", MEAN2,"    (5) Std Dev",STD2,"     (6) Percent",PERCENT2
  write(unit = 6, fmt= "(a)")"========================================"

  write(unit = 7, fmt= "(a)")"----------------------------------------"
  write(unit = 7, fmt= "(a)")"STATISTICS FOR Roptimal - Rgittins "
  write(unit = 7, fmt= "(a)")"----------------------------------------"
  write(unit = 7, fmt= "(a,i2,a,i2,a,f6.4)")
  "speed1 = ",speed1," speed2 = " ,speed2," alpha = " ,alpha
  write(unit = 7, fmt= "(a)")"----------------------------------------"
  write(unit = 7, fmt= "(a,i3)")" THE NUMBER OF SIMULATIONS IS  ", SIM
  write(unit = 7, fmt= "(a,f12.6,a,f12.6,a,f12.6)")
  "(1) Minimum", SMALL2,"    (2) Median ",MEDIAN2,"     (3) Maximum",BIG2
  write(unit = 7, fmt= "(a,f12.6,a,f12.6,a,f12.6)")
  "(4) Mean   ", MEAN2,"    (5) Std Dev",STD2,"     (6) Percent",PERCENT2
  write(unit = 7, fmt= "(a)")"========================================"
!-------------------------------------------------------------------------
  ORDER3 = 1.0e+20
  do j = 1,SIM
    do i = 1,SIM
      if(ORDER3(j) >= Subopt3(i)) then
        ORDER3(j) = Subopt3(i)
       k = i
      end if
```

```
    end do
     Subopt3(k) = 1.0e+21
   end do

   j = 0
   do i = 1,SIM
     if(ORDER3(i) > 10e-8) j = j + 1
   end do
  if (mod(j,2) == 1 .and. j > 0)then
    MEDIAN3 = ORDER3(SIM - j + ((j+1)/2))
   else if (mod(j,2) == 0 .and. j > 0)then
     MEDIAN3 = (ORDER3(SIM - j + (j/2)) + ORDER3(SIM - j + ((j/2)+1)))/2d0
   else if (j == 0) then
     MEDIAN3 = 0d0
   end if

   sum = 0d0
   sum2 = 0d0
   do i = 1,SIM
     sum = sum + ORDER3(i)
     sum2 = sum2 + ((ORDER3(i))**2)
   end do

   MEAN3 = (sum/SIM*1d0)
   t3 = (((1d0/SIM*1d0)*sum2)) - (MEAN3**2)

   if(t3 > 0)then
     STD3 = sqrt((((1d0/SIM*1d0)*sum2)) - ((MEAN3)**2))
   else
     STD3 = 0d0
   end if
   PERCENT3 = 100.0d0 - (100.0d0*j/SIM*1.0d0)

   write(unit = 6, fmt= "(a)")"----------------------------------------"
   write(unit = 6, fmt= "(a)")"STATISTICS FOR Rpolicy - Rgittins "
   write(unit = 6, fmt= "(a)")"----------------------------------------"
   write(unit = 6, fmt= "(a,i2,a,i2,a,f6.4)")
   "speed1 = ",speed1," speed2 = " ,speed2," alpha = " ,alpha
   write(unit = 6, fmt= "(a)")"----------------------------------------"
   write(unit = 6, fmt= "(a,i3)")" THE NUMBER OF SIMULATIONS IS  ", SIM
   write(unit = 6, fmt= "(a,f12.6,a,f12.6,a,f12.6)")
   "(1) Minimum", SMALL3,"    (2) Median ",MEDIAN3,"     (3) Maximum",BIG3
   write(unit = 6, fmt= "(a,f12.6,a,f12.6,a,f12.6)")
   "(4) Mean   ", MEAN3,"    (5) Std Dev",STD3,"     (6) Percent",PERCENT3
   write(unit = 6, fmt= "(a)")"========================================="

   write(unit = 7, fmt= "(a)")"----------------------------------------"
   write(unit = 7, fmt= "(a)")"STATISTICS FOR Rpolicy - Rgittins "
   write(unit = 7, fmt= "(a)")"----------------------------------------"
   write(unit = 7, fmt= "(a,i2,a,i2,a,f6.4)")
   "speed1 = ",speed1," speed2 = " ,speed2," alpha = " ,alpha
   write(unit = 7, fmt= "(a)")"----------------------------------------"
   write(unit = 7, fmt= "(a,i3)")" THE NUMBER OF SIMULATIONS IS  ", SIM
   write(unit = 7, fmt= "(a,f12.6,a,f12.6,a,f12.6)")
   "(1) Minimum", SMALL3,"    (2) Median ",MEDIAN3,"     (3) Maximum",BIG3
   write(unit = 7, fmt= "(a,f12.6,a,f12.6,a,f12.6)")
   "(4) Mean   ", MEAN3,"    (5) Std Dev",STD3,"     (6) Percent",PERCENT3
   write(unit = 7, fmt= "(a)")"========================================="

end do
```

```
end program
!=====================================================================
subroutine GETsubset(E,argmax,k,h)
implicit none
integer :: k,h,E
integer, dimension(4) :: argmax

h = h - 2**(E - argmax(k - 1))
return
end
!=====================================================================
subroutine GETAmatrix_LIMIT(AV,S,count,countcomp,P,h,A)
implicit none
integer :: d,g,i,j,h,f,k,count,countcomp,E
double precision :: matsum,Z,sum
integer, dimension(15,4) :: S
double precision, dimension(4,4) :: P
double precision, dimension(15,4) :: A,AV
double precision, dimension(countcomp,countcomp) :: Mmat,Pmat
double precision, dimension(countcomp,1) :: Pvec,Psum,Product
double precision, dimension(count) :: V1,V2

E = 4
!-----------------------------------------------
Mmat = 0d0
!-----------------------------------------------
d = 0
do i = 1,E
  if(S(h,i) == 0) then
    if(d < countcomp) d = d + 1
    Mmat(d,d) = 1 - P(i,i)
    g = d
    do j = 1,E
      if (i < j) then
        if (S(h,j) == 0) then
          if (g < countcomp) g = g + 1
          Mmat(d,g) = (-1.0d0)*P(i,j)
          Mmat(g,d) = (-1.0d0)*P(j,i)
        end if
      end if
    end do
  end if
end do
!-----------------------------------------------
Psum = 0d0
Pvec = 0d0
Pmat = 0d0
d = 0
do i = 1,E
  g = 0
  if (S(h,i) == 0) then
    if (d < countcomp) d = d + 1
    do j = 1,E
      if (S(h,j) == 0) then
        if(g < countcomp) g = g + 1
        Pmat(d,g) = P(i,j)
      end if
    end do
  end if
```

```
end do
!-------------------------------------------------
do i = 1,countcomp
  do j = 1,countcomp
    Psum(i,1) = Psum(i,1) + Pmat(i,j)
  end do
end do do i = 1,countcomp
  Pvec(i,1) = (1 - Psum(i,1))
end do
!-------------------------------------------------
Pvec = 1 + Pvec
!-------------------------------------------------
call GETinverse(countcomp,Mmat)
Product = 0d0
do i = 1,countcomp
  sum = 0d0
  do j = 1,countcomp
    sum = sum + (Mmat(i,j)*Pvec(j,1))
  end do
  Product(i,1) = sum
end do
Pvec = Product
!-------------------------------------------------
f = 0
do k = 1,4
  if (S(h,k) == 0) then
    if (f < countcomp) f = f + 1
    AV(h,k) = Pvec(f,1)
  end if
end do

d = 0
do i = 1,E
  matsum = 0
  if (S(h,i) == 1) then
    if (d < count) d = d + 1
    do j = 1,E
      if (S(h,j) == 0) matsum = matsum + (P(i,j) * AV(h,j))
    end do
    V1(d) = matsum
    matsum = 0
    do j = 1,E
      if (S(h,j) == 1) matsum = matsum + (P(i,j))
    end do
    V2(d) = matsum
    AV(h,i) = V1(d) + V2(d)
  end if
end do

AV(15,:) = (/1,1,1,1/)

f = 0
do k = 1,4
  if (S(h,k) == 1) then
    if (f < count) f = f + 1
    A(h,k) = AV(h,k)
  end if
end do
return
end
```

```
!=====================================================================
subroutine GETindices(b,k,S,h,r,A,argmax,G,IND,INDEX)
implicit none
integer :: b,i,h,k,l,j,E
double precision :: max,sum
integer, dimension(4) :: argmax
double precision, dimension(4) :: r,G,IND
double precision, dimension(15,4) :: A
integer, dimension(15,4) :: S
double precision, dimension(4,4) :: INDEX

if (k == 1) then
  max = -1000000
  do i = 1,4
    if (r(i)/A(h,i) > max) then
      max = r(i)/A(h,i)
      argmax(1) = i
    end if
  end do
  G(argmax(1)) = max
  IND(argmax(1)) = G(argmax(1))
else
  E = 4
  max = -1000000
  do i = 1,4
    l = 15
    sum = 0
    if (S(h,i) == 1) then
      do j = 1,k-1
        sum = sum + A(l,i)*G(argmax(j))
        l = l - 2**(E - argmax(j))
      end do
      if ((r(i) - sum)/A(h,i) > max) then
        max = (r(i) - sum)/A(h,i)
        argmax(k) = i
      end if
    end if
  end do
  G(argmax(k)) = max
  IND(argmax(k)) = G(argmax(k)) + IND(argmax(k - 1))
end if

INDEX(b,argmax(k)) = IND(argmax(k))
return
end
!=====================================================================
subroutine GETinverse(countcomp,Mmat)
implicit none
integer :: i,j,countcomp
double precision :: a,b,c,d,determinant
double precision, dimension(countcomp,countcomp) :: Mmat
double precision, dimension(3,3) :: Cofactor

if (countcomp == 1) then
  Mmat(1,1) = 1d0/Mmat(1,1)
else if (countcomp == 2) then
  a = Mmat(1,1)
  b = Mmat(1,2)
  c = Mmat(2,1)
  d = Mmat(2,2)
```

```
  determinant = (a*d) - (b*c)
  Mmat(1,1) = d/determinant
  Mmat(1,2) = -1d0*(b/determinant)
  Mmat(2,1) = -1d0*(c/determinant)
  Mmat(2,2) = a/determinant
else if (countcomp == 3) then
  Cofactor(1,1) = ((Mmat(2,2)*Mmat(3,3)) - (Mmat(2,3)*Mmat(3,2)))
  Cofactor(1,2) = -1d0*((Mmat(2,1)*Mmat(3,3)) - (Mmat(2,3)*Mmat(3,1)))
  Cofactor(1,3) = ((Mmat(2,1)*Mmat(3,2)) - (Mmat(2,2)*Mmat(3,1)))

  Cofactor(2,1) = -1d0*((Mmat(1,2)*Mmat(3,3)) - (Mmat(1,3)*Mmat(3,2)))
  Cofactor(2,2) = ((Mmat(1,1)*Mmat(3,3)) - (Mmat(1,3)*Mmat(3,1)))
  Cofactor(2,3) = -1d0*((Mmat(1,1)*Mmat(3,2)) - (Mmat(1,2)*Mmat(3,1)))

  Cofactor(3,1) = ((Mmat(1,2)*Mmat(2,3)) - (Mmat(1,3)*Mmat(2,2)))
  Cofactor(3,2) = -1d0*((Mmat(1,1)*Mmat(2,3)) - (Mmat(1,3)*Mmat(2,1)))
  Cofactor(3,3) = ((Mmat(1,1)*Mmat(2,2)) - (Mmat(1,2)*Mmat(2,1)))

  determinant = Mmat(1,1)*Cofactor(1,1) + Mmat(1,2)*Cofactor(1,2)
              + Mmat(1,3)*Cofactor(1,3)

  do i = 1,3
    do j = 1,3
      Mmat(i,j) = Cofactor(j,i)/determinant
    end do
  end do
end if
return
end
!=============================================================
subroutine GETPmatrix(seed,Pmat)
implicit none
integer :: i,j,k,a
integer, dimension(3) :: seed
double precision ::UNIFORM,sum,L,U
double precision, dimension(4,4,4) :: PMat

do k = 1,2
  do i = 1,4
    if(i == 1) then
    PMat(k,i,1) = 0.0d0
    else if(i > 1) then
      L = 0.00001
      U = 0.00002
      call GETuniform(seed,L,U,UNIFORM)
      PMat(k,i,1) = UNIFORM
    end if
    do j = 2,4
      L = 0.1
      U = 0.9
      call GETuniform(seed,L,U,UNIFORM)
      PMat(k,i,j) = UNIFORM
    end do
  end do
end do

do k = 1,2
  do i = 1,4
    sum = 0d0
    do j = 1,4
```

```
      sum = sum + PMat(k,i,j)
    end do
    do j = 1,4
      PMat(k,i,j) = Pmat(k,i,j)*(1.0d0/sum)
    end do
  end do
end do

L = 0.1
U = 0.9

  do k = 3,4
    do i = 1,4
      do j = 1,4
        call GETuniform(seed,L,U,UNIFORM)
        PMat(k,i,j) = UNIFORM
      end do
    end do
  end do

  do k = 3,4
    do i = 1,4
      sum = 0d0
      do j = 1,4
        sum = sum + PMat(k,i,j)
      end do
      do j = 1,4
        PMat(k,i,j) = Pmat(k,i,j)*(1.0d0/sum)
      end do
    end do
  end do
return end
!=====================================================================
subroutine GETRmatrix(seed,RMat)
implicit none
integer :: i,j
integer, dimension(3) :: seed
double precision :: L,U,x double
precision, dimension(4,4) :: RMat

do i = 1,2
 if(i == 1) then
   L = 7.5d0
   U = 8.0d0
 else if(i == 2) then
   L = 6.5d0
   U = 7.0d0
 end if
 call GETuniform(seed,L,U,x)
 RMat(i,1) = x
end do

L = 0.00001
U = 0.00002
do i = 1,2
  do j = 2,4
    call GETuniform(seed,L,U,x)
    RMat(i,j) = x
  end do
end do
```

```
do i = 3,4
 if(i == 3) then
   L = 5.0d0
   U = 5.5d0
 else if(i == 4) then
   L = 4.0d0
   U = 4.5d0
 end if
   do j = 1,4
     call GETuniform(seed,L,U,x)
     RMat(i,j) = x
   end do
end do
return
end
!=====================================================================
subroutine GETuniform(seed,L,U,x)
implicit none
integer, dimension(3) :: seed
double precision :: r,s,x,L,U

seed(1) = mod(171*seed(1),30269)
seed(2) = mod(172*seed(2),30307)
seed(3) = mod(170*seed(3),30323)

s = seed(1)*1d0/30269 + seed(2)*1d0/30307 + seed(3)*1d0/30323
r = s - int(s)

x = L + (U-L)*r
return
end
!=====================================================================
subroutine GETrandom(seed,x)
implicit none
integer :: x
integer, dimension(3) :: seed
double precision :: r,s

seed(1) = mod(171*seed(1),30269)
seed(2) = mod(172*seed(2),30307)
seed(3) = mod(170*seed(3),30323)

s = seed(1)*1d0/30269 + seed(2)*1d0/30307 + seed(3)*1d0/30323
r = s - int(s)

x = int(16*r)
return
end
!=====================================================================
subroutine Check(IND,num)
implicit none
integer :: i,j,k,c,d,num
double precision :: max
integer, dimension(4,4) :: ORDER1,ORDER2
double precision, dimension(4,4) :: IND

c = 0
d = 0
ORDER1 = 0
```

```
do k = 1,16
  max = -100000
  do i = 1,4
    do j = 1,4
      if (IND(i,j) > max) then
        max = IND(i,j)
        c = i
        d = j
      end if
    end do
  end do
  ORDER1(c,d) = k
  IND(c,d) = 0
end do

c = 0
d = 0
ORDER2 = 0
do k = 1,16
  max = -100000
  do i = 1,4
    do j = 1,4
      if (IND(i,j) > max) then
        max = IND(i,j)
        c = i
        d = j
      end if
    end do
  end do
  ORDER2(c,d) = k
  IND(c,d) = 0
end do

num = 0
do i = 1,4
  do j = 1,4
    if(ORDER1(i,j) /= ORDER2(i,j)) then
      num = num + 1
    end if
  end do
end do
return
end
!=====================================================================
subroutine GETbandit1(IND,bandit1,cond)
implicit none
integer :: i,j,c,cond,bandit1
double precision :: max
double precision, dimension(3) :: Gsub
double precision, dimension(4) :: min
double precision, dimension(4,4) :: IND

min = 0d0
do i= 1,4
  min(i) = 1000000d0
  do j = 1,4
    if (IND(i,j) < min(i)) then
      min(i) = IND(i,j)
    end if
  end do
```

```
end do

do i = 1,3
  max = -10000
  do j = 1,4
    if (min(j) > max) then
      max = min(j)
      Gsub(i) = min(j)
      c = j
    end if
  end do
  if (i == 1) then
    bandit1 = c
  end if
  min(c) = -100000
end do

if (Gsub(1) /= Gsub(2) .and. Gsub(2) /= Gsub(3) .and. Gsub(1) /= Gsub(3)) then
  cond = 1
else
  cond = 2
end if
return
end
!=====================================================================
subroutine GETrewards
(c,Rmat,Pmat,INDEX,alpha,speed1,speed2,bandit1,Rewoptimal,Rewpolicy,Rewgittins)
implicit none
integer :: a,b,e,f,h,i,j,k,l,m,n,q,speed1,speed2,bandit1,bandit2,bplace
integer :: b1,b2
integer,dimension(3) :: v,uv,vu,cnew
integer,dimension(3) :: stind,bind
integer,dimension(4) :: c,IS,u
integer,dimension(4) :: st1,st2
double precision :: alpha,Rmn,Rnm,maxi,Rewoptimal,Rewpolicy
double precision :: R12,R21,Rewgittins
double precision, dimension(2) :: dis
double precision, dimension(4) :: d
double precision, dimension(6) :: max,rew
double precision, dimension(8) :: Psum
double precision, dimension(10) :: PSumind
double precision, dimension(4,2) :: BP1,POL1
double precision, dimension(4,4) :: Rmat
double precision, dimension(4,4) :: INDEX
double precision, dimension(4,4,4) :: Pmat
double precision, dimension(4,4,4,2) :: BP2,POL2
double precision, dimension(4,4,4,4,2) :: O0,BP
double precision, dimension(4,4,4,4,2) :: I0,I1,I2
double precision, dimension(4,4,4,4,4,2) :: O3,O4,O6,O8,O9
double precision, dimension(4,4,4,4,4,2) :: I3,I4,I6,I8,I9
double precision, dimension(4,4,4,4,4,4,2) :: O1,O2,O5,O7,O10,O11
double precision, dimension(4,4,4,4,4,4,2) :: I5,I7,I10,I11

d(1) = (1.0d0/alpha)*(1.0d0 - exp(-1.0d0*(alpha/(1.0d0*speed1))))
d(2) = (1.0d0/alpha)*(1.0d0 - exp(-1.0d0*(alpha/(1.0d0*speed2))))
d(3) = (1.0d0/alpha)*(1.0d0 - exp(-1.0d0*(alpha/12.0d0)))
d(4) = exp(-1.0d0*(alpha/12.0d0))

dis(1) = exp(-1.0d0*(alpha/(1.0d0*speed1)))
dis(2) = exp(-1.0d0*(alpha/(1.0d0*speed2)))
```

```
O0=0d0
O1=0d0
O2=0d0
O3=0d0
O4=0d0
O5=0d0
O6=0d0
O7=0d0
O8=0d0
O9=0d0
O10=0d0
O11=0d0
BP1=0d0
BP2=0d0
BP=0d0
POL1 = 0d0
POL2 = 0d0
I0=0d0
I1=0d0
I2=0d0
I3=0d0
I4=0d0
I5=0d0
I6=0d0
I7=0d0
I8=0d0
I9=0d0
I10=0d0
I11=0d0

do q = 1,int((1.0d0*speed1*speed2)*(-1.0d0/alpha)
        *log((alpha*10e-8)/((5.0d0/d(1))+(5.0d0/d(2)))))+1
  if (mod(q,2) == 1) then
    e = 1
    f = 2
  else if (mod(q,2) == 0) then
    e = 2
    f = 1
  end if
  do i = 1,4
    do j = 1,4
      do k = 1,4
        do l = 1,4
          IS = (/i,j,k,l/)
          max(1) = -10000d0
          rew = 0d0
          do m = 1,4
            do a = 2,6
              max(a) = -10000d0
            end do
!---------------------------
            if (m == bandit1) then
              h = 0
              Psum(7) = 0d0
              do a = 1,4
                if (a /= bandit1) then
                  h = h + 1
                  uv(h) = IS(a)
                  vu(h) = a
```

```
                end if
                Psum(7) = Psum(7) + Pmat(m,IS(m),a)*BP1(a,e)
              end do
BP1(IS(bandit1),f) = Rmat(bandit1,IS(bandit1)) + dis(1)*Psum(7)
              maxi = -10000d0
              bandit2 = 0
              bplace = 0
              do b = 1,3
                if (INDEX(vu(b),uv(b)) > maxi) then
                  maxi = INDEX(vu(b),uv(b))
                  bandit2 = vu(b)
                  bplace = b
                end if
              end do
            end if
            h = 0
!---------------------------
            do n = 1,4
              if (m /= n) then
!---------------------------
                h = h + 1
!---------------------------
Rmn = ((d(3)*Rmat(m,IS(m))/d(1)) + (d(3)*Rmat(n,IS(n))/d(2)))
Rnm = ((Rmat(n,IS(n))/d(1)) + (Rmat(m,IS(m))/d(2)))*d(3)
                do b = 1,8
                  PSum(b) = 0d0
                end do
                do a = 1,4
!---------------------------
                  if (m == bandit1 .and. n == bandit2) then
                    v = uv
                    v(bplace) = a
Psum(8) = Psum(8) + Pmat(bandit2,IS(bandit2),a)*BP2(v(1),v(2),v(3),e)
                  end if
!---------------------------
                  u = IS
                  u(m) = a
Psum(1) = Psum(1) + Pmat(m,IS(m),a)*03(u(1),u(2),u(3),u(4),n,e)
Psum(2) = Psum(2) + Pmat(m,IS(m),a)*04(u(1),u(2),u(3),u(4),n,e)
Psum(3) = Psum(3) + Pmat(m,IS(m),a)*06(u(1),u(2),u(3),u(4),n,e)
Psum(4) = Psum(4) + Pmat(m,IS(m),a)*08(u(1),u(2),u(3),u(4),n,e)
Psum(5) = Psum(5) + Pmat(m,IS(m),a)*09(u(1),u(2),u(3),u(4),n,e)
                  do b = 1,4
                    u(n) = b
Psum(6) = Psum(6) + Pmat(n,IS(n),b)*Pmat(m,IS(m),a)*00(u(1),u(2),u(3),u(4),e)
                  end do
                end do
rew(1) = Rmn + d(4)*01(i,j,k,l,m,n,e)
rew(2) = Rnm + d(4)*Psum(2)
rew(3) = Rmn + d(4)*05(i,j,k,l,m,n,e)
rew(4) = Rnm + d(4)*07(i,j,k,l,n,m,e)
rew(5) = Rmn + d(4)*Psum(5)
rew(6) = Rnm + d(4)*010(i,j,k,l,n,m,e)
!-----------------------
                if (m == bandit1 .and. n == bandit2) then
BP2(uv(1),uv(2),uv(3),f) = Rmat(bandit2,IS(bandit2)) + dis(2)*Psum(8)
                end if
!-----------------------
                do a = 1,6
                  if (rew(a) > max(a)) then
```

```
                  max(a) = rew(a)
                end if
              end do
O1(i,j,k,l,m,n,f) = Rmn + d(4)*O2(i,j,k,l,m,n,e)
O2(i,j,k,l,m,n,f) = Rmn + d(4)*Psum(1)
O5(i,j,k,l,m,n,f) = Rmn + d(4)*Psum(3)
O7(i,j,k,l,n,m,f) = Rnm + d(4)*Psum(4)
O10(i,j,k,l,m,n,f) = Rmn + d(4)*O11(i,j,k,l,m,n,e)
O11(i,j,k,l,m,n,f) = Rmn + d(4)*Psum(6)
            end if
          end do
O3(i,j,k,l,m,f) = max(2)
O4(i,j,k,l,m,f) = max(3)
O6(i,j,k,l,m,f) = max(4)
O8(i,j,k,l,m,f) = max(5)
O9(i,j,k,l,m,f) = max(6)
        end do
O0(i,j,k,l,f) = max(1)
!-------------------------------------------------------------------------
!                          GITTINS INDICES BIT
!-------------------------------------------------------------------------
        maxi = -10000d0
        do a = 1,4
          if(INDEX(a,IS(a)) > maxi) then
            maxi = INDEX(a,IS(a))
            b1 = a
          end if
        end do
        h = 0
        do a = 1,4
          if(a /= b1) then
            h = h + 1
            stind(h) = IS(a)
            bind(h) = a
          end if
        end do
        maxi = -10000d0
        do a = 1,3
          if(INDEX(bind(a),stind(a)) > maxi ) then
            maxi = INDEX(bind(a),stind(a))
            b2 = bind(a)
          end if
        end do
R12 = Rmat(b1,IS(b1))*d(3)/d(1) + Rmat(b2,IS(b2))*d(3)/d(2)
R21 = Rmat(b2,IS(b2))*d(3)/d(1) + Rmat(b1,IS(b1))*d(3)/d(2)
        PSumind = 0d0
        do a = 1,4
          st1 = IS
          st1(b1) = a
PSumind(1) = PSumind(1) + Pmat(b1,IS(b1),a)
          *I3(st1(1),st1(2),st1(3),st1(4),b2,e)
PSumind(2) = PSumind(2) + Pmat(b1,IS(b1),a)
          *I4(st1(1),st1(2),st1(3),st1(4),b2,e)
PSumind(4) = PSumind(4) + Pmat(b1,IS(b1),a)
          *I6(st1(1),st1(2),st1(3),st1(4),b2,e)
PSumind(6) = PSumind(6) + Pmat(b1,IS(b1),a)
          *I8(st1(1),st1(2),st1(3),st1(4),b2,e)
PSumind(8) = PSumind(8) + Pmat(b1,IS(b1),a)
          *I9(st1(1),st1(2),st1(3),st1(4),b2,e)
          st2 = IS
```

```
            st2(b2) = a
PSumind(3) = PSumind(3) + Pmat(b2,IS(b2),a)
            *I4(st2(1),st2(2),st2(3),st2(4),b1,e)
PSumind(5) = PSumind(5) + Pmat(b2,IS(b2),a)
            *I6(st2(1),st2(2),st2(3),st2(4),b1,e)
PSumind(7) = PSumind(7) + Pmat(b2,IS(b2),a)
            *I8(st2(1),st2(2),st2(3),st2(4),b1,e)
PSumind(9) = PSumind(9) + Pmat(b2,IS(b2),a)
            *I9(st2(1),st2(2),st2(3),st2(4),b1,e)
            do b = 1,4
              st1(b2) = b
PSumind(10) = PSumind(10) + Pmat(b1,IS(b1),a)*Pmat(b2,IS(b2),b)
             *I0(st1(1),st1(2),st1(3),st1(4),e)
            end do
          end do
I0(i,j,k,l,f) = R12 + d(4)*I1(i,j,k,l,e)
I1(i,j,k,l,f) = R12 + d(4)*I2(i,j,k,l,e)
I2(i,j,k,l,f) = R12 + d(4)*PSumind(1)
I3(i,j,k,l,b1,f) = R21 + d(4)*PSumind(2)
I3(i,j,k,l,b2,f) = R12 + d(4)*PSumind(3)
I4(i,j,k,l,b1,f) = R12 + d(4)*I5(i,j,k,l,b1,b2,e)
I4(i,j,k,l,b2,f) = R21 + d(4)*I5(i,j,k,l,b2,b1,e)
I5(i,j,k,l,b1,b2,f) = R12 + d(4)*PSumind(4)
I5(i,j,k,l,b2,b1,f) = R21 + d(4)*PSumind(5)
I6(i,j,k,l,b1,f) = R21 + d(4)*I7(i,j,k,l,b2,b1,e)
I6(i,j,k,l,b2,f) = R12 + d(4)*I7(i,j,k,l,b1,b2,e)
I7(i,j,k,l,b1,b2,f) = R12 + d(4)*PSumind(7)
I7(i,j,k,l,b2,b1,f) = R21 + d(4)*PSumind(6)
I8(i,j,k,l,b1,f) = R12 + d(4)*PSumind(8)
I8(i,j,k,l,b2,f) = R21 + d(4)*PSumind(9)
I9(i,j,k,l,b1,f) = R21 + d(4)*I10(i,j,k,l,b2,b1,e)
I9(i,j,k,l,b2,f) = R12 + d(4)*I10(i,j,k,l,b1,b2,e)
I10(i,j,k,l,b1,b2,f) = R12 + d(4)*I11(i,j,k,l,b1,b2,e)
I10(i,j,k,l,b2,b1,f) = R21 + d(4)*I11(i,j,k,l,b2,b1,e)
I11(i,j,k,l,b1,b2,f) = R12 + d(4)*PSumind(10)
I11(i,j,k,l,b2,b1,f) = R21 + d(4)*PSumind(10)
!-----------------------------------------------------------------------
        end do
      end do
    end do
  end do

  if(q == int((speed2*1d0)*(-1.0d0/alpha)
        *log((alpha*10e-8)/(5.0d0/d(2))))+1) then
  POL2 = BP2
  end if
  if(q == int((speed1*1d0)*(-1.0d0/alpha)
        *log((alpha*10e-8)/(5.0d0/d(1))))+1) then
  POL1 = BP1
  end if

end do
Rewoptimal = O0(c(1),c(2),c(3),c(4),f)
Rewgittins = I0(c(1),c(2),c(3),c(4),f)

h = 0
do a = 1,4
  if (a /= bandit1) then
    h = h + 1
    cnew(h) = c(a)
```

```
    v(h) = a
  end if
end do
Rewpolicy = POL1(c(bandit1),f) + POL2(cnew(1),cnew(2),cnew(3),f)
return
end
!=====================================================================
```

Appendix H

```
program stochastic_branching_bandit

implicit none integer :: i,j,k,h,d,M,SIM,y,c integer, dimension(3)
:: seed integer, dimension(2) :: argmax,nmax,STATE integer,
dimension(3,2) :: S double precision :: alpha,Z,t,Prep,Pbd double
precision, dimension(2) :: Parr,G,INDEX
double precision, dimension(3) :: RMat,Ropt,Rmin,Rmax,SUMROPT
double precision, dimension(2,3) :: SMALL,BIG
double precision, dimension(3,3) :: Pint
double precision, dimension(3,2) :: A,AV
double precision, dimension(2,3,500) :: Subopt

SIM = 500
seed = (/22198,16952,5217/)

open(unit=7,file="BRANCHING_STATS.dat")
write(unit=7,fmt="(a)")""

write(unit=6,fmt="(a)")""

do j = 3,1,-1
  d = j
  do i = 1,0,-1
    t = d/2**i
    if (t < 1) S(j,(2-i)) = 0
    if (t>= 1) then
      S(j,(2-i)) = 1
      d = mod(d,2**i)
    end if
  end do
end do

do c = 1,5
  if (c == 1) then
    alpha = 1.6d0
    nmax = (/25,25/)
  else if(c == 2) then
    alpha = 0.8d0
    nmax = (/75,75/)
  else if (c == 3) then
    alpha = 0.4d0
    nmax = (/100,100/)
  else if (c == 4) then
    alpha = 0.2d0
    nmax = (/200,200/)
  else if (c == 5) then
    alpha = 0.1d0
    nmax = (/300,300/)
```

```
  end if

  SMALL = 1.0e+20
  BIG = -1*(1.0e+20)
  SUMROPT = 0.0d0

do y = 1,SIM

  Ropt = 0.0d0
  Rmax = 0.0d0
  Rmin = 0.0d0

  call GETSTATE(seed,STATE)
  call GETmachine(seed,M)
  M = M + 1
  call GETParrivals(seed,Parr)
  call GETPinternal(seed,Pint)
  call GETPrep_Pbd(seed,Prep,Pbd)
  call GETRmatrix(seed,Rmat)

  Z = exp(-1.0d0*alpha)
  h = 3
  G = 0
  INDEX = 0
  argmax = 0
  A = 0
  A(3,:) = (/1,1/)
  AV = 0
  do k = 1,2
    if (k > 1) call GETsubset(argmax,k,h)
    if (k > 1) call GETAmatrix(AV,S,Parr,Pint,Z,h,A)
    call GETindices(k,S,h,Rmat,A,argmax,G,INDEX)
  end do

  call GETrewards
  (SIM,nmax,M,STATE,Rmat,Parr,Pint,Prep,Pbd,INDEX,alpha,Z,Ropt,Rmax,Rmin)

  call GETsubopt(y,SIM,BIG,SMALL,Ropt,Rmax,Rmin,Subopt)

  do i = 1,3
    SUMROPT(i) = SUMROPT(i) + Ropt(i)
  end do

end do

SUMROPT = SUMROPT/SIM

call GETstats(SIM,BIG,SMALL,Subopt,SUMROPT)

end do

end program
!=============================================================
subroutine GETSTATE(seed,STATE)
implicit none
integer :: i,j,h,RANDOM
integer, dimension(3) :: seed
integer, dimension(2) :: STATE

call GETrandom(seed,RANDOM)
```

```
h = 0
do i = 2,6
  do j = 2,6
    h = h+1
    if(h == RANDOM) STATE = (/i,j/)
  end do
end do
return
end
!=============================================================
subroutine GETsubset(argmax,k,h)
implicit none
integer :: k,h
integer, dimension(2) :: argmax

h = h - 2**(2 - argmax(k - 1))
return
end
!=============================================================
subroutine GETAmatrix(AV,S,Parr,Pint,Z,h,A)
implicit none
integer :: d,i,j,h,NUM,arr,int
integer, dimension(3,2) :: S
integer, dimension(4,2) :: S2
double precision :: Z,t,SUMint,SUMarr
double precision, dimension(2) :: Parr,P
double precision, dimension(3) :: ET
double precision, dimension(3,3) :: Pint
double precision, dimension(3,2) :: A,AV

do j = 3,1,-1
  d = j
  do i = 1,0,-1
    t = d/2**i
    if (t < 1) S2(j,(2-i)) = 0
    if (t>= 1) then
      S2(j,(2-i)) = 1
      d = mod(d,2**i)
    end if
  end do
end do
S2(4,:) = (/0,0/)
do NUM = 1,10000
  do i = 1,2
    if(S(h,i) == 1) then
      ET(i) = AV(h,i)
    else if (S(h,i) == 0) then
      ET(i) = 1
    end if
  end do
  ET(3) = 1
  SUMarr = 0.0d0
  do arr = 1,4
    P = 0.0d0
    do i = 1,2
      if(S2(arr,i) == 1) then
        P(i) = Parr(i)*ET(i)
      else if(S2(arr,i) == 0) then
        P(i) = 1 - Parr(i)
      end if
```

```
    end do
      SUMarr = SUMarr + P(1)*P(2)*Z
  end do
  do i = 1,2
    if(S(h,i) == 1) then
      SUMint = 0.0d0
      do int = 1,3
        SUMint = SUMint + Pint(i,int)*ET(int)*SUMarr
      end do
      AV(h,i) = SUMint
    end if
  end do
end do
AV(3,:) = (/Z,Z/)
do i = 1,2
  if (S(h,i) == 1) then
    A(h,i) = (1 - AV(h,i))/(1 - Z)
  end if
end do
return
end
!=============================================================
subroutine GETindices(k,S,h,r,A,argmax,G,IND)
implicit none
integer :: i,h,k,l,j,E
double precision :: maxi,sum
integer, dimension(2) :: argmax
double precision, dimension(2) :: r,G,IND
double precision, dimension(3,2) :: A
integer, dimension(3,2) :: S

if (k == 1) then
  maxi = -1000000
  do i = 1,2
    if (r(i)/A(h,i) > maxi) then
      maxi = r(i)/A(h,i)
      argmax(1) = i
    end if
  end do
  G(argmax(1)) = maxi
  IND(argmax(1)) = G(argmax(1))
else
  E = 2
  maxi = -1000000
  do i = 1,2
    l = 3
    sum = 0
    if (S(h,i) == 1) then
      do j = 1,k-1
        sum = sum + A(l,i)*G(argmax(j))
        l = l - 2**(E - argmax(j))
      end do
      if ((r(i) - sum)/A(h,i) > maxi) then
        maxi = (r(i) - sum)/A(h,i)
        argmax(k) = i
      end if
    end if
  end do
  G(argmax(k)) = maxi
  IND(argmax(k)) = G(argmax(k)) + IND(argmax(k - 1))
```

```
end if
return
end
!=============================================================
subroutine GETParrivals(seed,Parr)
implicit none
integer :: i
integer, dimension(3) :: seed
double precision :: UNIFORM,sum,L,U
double precision, dimension(2) :: Parr

L = 0.01
U = 0.99

do i = 1,2
  call GETuniform(seed,L,U,UNIFORM)
    Parr(i) = UNIFORM
end do
return
end
!=============================================================
subroutine GETPinternal(seed,Pint)
implicit none
integer :: i,j
integer, dimension(3) :: seed
double precision :: UNIFORM,sum,L,U
double precision, dimension(3,3) :: Pint

L = 0.01
U = 0.99

Pint = 0.0d0

do i = 1,2
  do j = 1,3
    call GETuniform(seed,L,U,UNIFORM)
    Pint(i,j) = UNIFORM
  end do
end do

do i = 1,2
  sum = 0d0
  do j = 1,3
    sum = sum + Pint(i,j)
  end do
  do j = 1,3
    Pint(i,j) = Pint(i,j)*(1.0d0/sum)
  end do
end do

do i = 1,3
  Pint(3,i) = 1.0d0
end do
return
end
!=============================================================
subroutine GETPrep_Pbd(seed,Prep,Pbd)
implicit none
integer, dimension(3) :: seed
double precision :: UNIFORM,L,U,Prep,Pbd
```

```
L = 0.50
U = 0.99
call GETuniform(seed,L,U,UNIFORM) Prep = UNIFORM

L = 0.01
U = 0.50
call GETuniform(seed,L,U,UNIFORM) Pbd = UNIFORM

return
end
!=============================================================
subroutine GETRmatrix(seed,Rmat)
implicit none
integer :: i
integer, dimension(3) :: seed
double precision :: L,U,x
double precision, dimension(3) :: RMat

L = 2d0
U = 5d0

do i = 1,2
  call GETuniform(seed,L,U,x)
  RMat(i) = x
end do

RMat(3) = 0.0d0
return
end
!=============================================================
subroutine GETuniform(seed,L,U,x)
implicit none
integer, dimension(3) :: seed
double precision :: r,s,x,L,U

seed(1) = mod(171*seed(1),30269)
seed(2) = mod(172*seed(2),30307)
seed(3) = mod(170*seed(3),30323)

s = seed(1)*1d0/30269 + seed(2)*1d0/30307 + seed(3)*1d0/30323
r = s - int(s)

x = L + (U-L)*r
return
end
!=============================================================
subroutine GETrandom(seed,x)
implicit none
integer :: x
integer, dimension(3) :: seed
double precision :: r,s

seed(1) = mod(171*seed(1),30269)
seed(2) = mod(172*seed(2),30307)
seed(3) = mod(170*seed(3),30323)

s = seed(1)*1d0/30269 + seed(2)*1d0/30307 + seed(3)*1d0/30323
r = s - int(s)
```

```
x = int(25*r)
return
end
!=============================================================
subroutine GETmachine(seed,x)
implicit none
integer :: x
integer, dimension(3) :: seed
double precision :: r,s

seed(1) = mod(171*seed(1),30269)
seed(2) = mod(172*seed(2),30307)
seed(3) = mod(170*seed(3),30323)

s = seed(1)*1d0/30269 + seed(2)*1d0/30307 + seed(3)*1d0/30323
r = s - int(s)

x = int(3*r)
return
end
!=============================================================
subroutine GETchoicemat(IS,choicemat)
implicit none
integer :: a,i,j,count
integer, dimension(2) :: IS,cvec
integer, dimension(3,3) :: choicemat

choicemat = 0
cvec = 0
count = 0
do i = 1,2
  if(IS(i) == 2) then
    cvec(i) = 1
    a = i
  else if(IS(i) > 2) then
    cvec(i) = 2
  end if
end do

do i = 1,2
  count = count + cvec(i)
end do

choicemat = 0

if(count == 0) then
  choicemat(3,3) = 1
else if(count == 1) then
  choicemat(a,3) = 1
else if(count >= 2) then
  do i = 1,2
    if(IS(i) >= 3) choicemat(i,i) = 1
    do j = 1,2
      if(i < j) then
        if(IS(i) >= 2 .and. IS(j) >= 2) choicemat(i,j) = 1
      end if
    end do
  end do
end if
return
```

```
end
!=============================================================
subroutine GETchoicevec(IS,choicevec)
implicit none
integer :: i
integer, dimension(2) :: IS
integer, dimension(3) :: choicevec

choicevec = 0
do i = 1,2
  if(IS(i) > 1) then
    choicevec(i) = 1
  end if
end do

if(IS(1) == 1 .and. IS(2) == 1) then
  choicevec(3) = 1
end if
return
end
!=============================================================
subroutine TWO_up(IS,nmax,choicemat,Parr,Pint,Pbd,RMat,Z,Vopt,e,f)
implicit none
integer :: a,b,i,j,d,e,f,ch1,ch2,int1,int2,arr1,arr2
integer, dimension(2) :: IS,n,nmax
integer, dimension(3,3) :: choicemat
integer, dimension(4,2) :: S
double precision :: Z,t,rew,maxi,PSum,Pbd,Parrivals,Pinternal
double precision, dimension(2) :: P1,P2,Parr
double precision, dimension(3) :: RMat
double precision, dimension(3,3) :: Pint
double precision, dimension(nmax(1)+1,nmax(2)+1,3,2) :: Vopt

do j = 3,1,-1
  d = j
  do i = 1,0,-1
    t = d/2**i
    if (t < 1) S(j,(2-i)) = 0
    if (t>= 1) then
      S(j,(2-i)) = 1
      d = mod(d,2**i)
    end if
  end do
end do
S(4,:) = (/0,0/)
maxi = - 10000.0d0
do ch1 = 1,3
  do ch2 = 1,3
    rew = 0.0d0
    PSum = 0.0d0
    if(ch1 <= ch2 .and. choicemat(ch1,ch2) == 1) then
      if(ch1 == 3) then
        a = 1
      else if(ch1 < 3) then
        a = 3
      end if
      if(ch2 == 3) then
        b = 1
      else if(ch2 < 3) then
        b = 3
```

```
      end if
      do int1 = 1,a
        do int2 = 1,b
          do arr1 = 1,4
            do arr2 = 1,4
              n = IS
              P1 = 0.0d0
              P2 = 0.0d0
              do i = 1,2
                if(S(arr1,i) == 1) then
                  n(i) = n(i) + 1
                  P1(i) = Parr(i)
                else if(S(arr1,i) == 0) then
                  P1(i) = 1 - Parr(i)
                end if
                if(S(arr2,i) == 1) then
                  n(i) = n(i) + 1
                  P2(i) = Parr(i)
                else if(S(arr2,i) == 0) then
                  P2(i) = 1 - Parr(i)
                end if
              end do
              if(ch1 < 3) n(ch1) = n(ch1) - 1
              if(ch2 < 3) n(ch2) = n(ch2) - 1
              if(int1 < 3 .and. ch1 < 3) n(int1) = n(int1) + 1
              if(int2 < 3 .and. ch2 < 3) n(int2) = n(int2) + 1
              if(n(1) <= nmax(1)+1.and.n(2) <= nmax(2)+1)then

Parrivals = P1(1)*P1(2)*P2(1)*P2(2)

Pinternal = Pint(ch1,int1)*Pint(ch2,int2)

PSum = PSum + ((1-Pbd)*(1-Pbd)*Parrivals*Pinternal
            *Z*Vopt(n(1),n(2),3,e)
            + 2*(Pbd*(1-Pbd)*Parrivals*Pinternal
            *Z*Vopt(n(1),n(2),2,e))&
            + Pbd*Pbd*Parrivals*Pinternal
            *Z*Vopt(n(1),n(2),1,e))
              end if
            end do
          end do
        end do
      end do
      rew = RMat(ch1) + RMat(ch2) + PSum
      if(rew > maxi) maxi = rew
    end if
  end do
end do

Vopt(IS(1),IS(2),3,f) = maxi return end
!=============================================================
subroutine
ONE_up(IS,nmax,choice,choicevec,Parr,Pint,Pbd,Prep,RMat,Z,Vopt,Vind,e,f)
implicit none
integer :: i,j,d,e,f,ch,int,arr,choice
integer, dimension(2) :: IS,n,nmax
integer, dimension(3) :: choicevec
integer, dimension(4,2) :: S
double precision :: Z,t,rew,maxi,PSum,PSumind,Parrivals,Pinternal,Pbd,Prep,Rewindex
double precision, dimension(2) :: P,Parr
```

```
double precision, dimension(3) :: RMat
double precision, dimension(3,3) :: Pint
double precision, dimension(nmax(1)+1,nmax(2)+1,2,2) :: Vind
double precision, dimension(nmax(1)+1,nmax(2)+1,3,2) :: Vopt

do j = 3,1,-1
  d = j
  do i = 1,0,-1
    t = d/2**i
    if (t < 1) S(j,(2-i)) = 0
    if (t>= 1) then
      S(j,(2-i)) = 1
      d = mod(d,2**i)
    end if
  end do
end do
S(4,:) = (/0,0/)
maxi = - 10000.0d0
Rewindex = 0.0d0
do ch = 1,3
  if(choicevec(ch) == 1) then
    rew = 0.0d0
    if(ch == choice) then
      Rewindex = 0.0d0
    end if
    if(ch == 3) then
      a = 1
    else if(ch < 3) then
      a = 3
    end if
    PSumind = 0.0d0
    PSum = 0.0d0
    do int = 1,a
      do arr = 1,4
        n = IS
        P = 0.0d0
        do i = 1,2
          if(S(arr,i) == 1) then
            n(i) = n(i) + 1
            P(i) = Parr(i)
          else if(S(arr,i) == 0) then
            P(i) = 1 - Parr(i)
          end if
        end do
        if(ch < 3) n(ch) = n(ch) - 1
        if(int < 3 .and. ch < 3) n(int) = n(int) + 1
        if(n(1) <= nmax(1)+1 .and. n(2) <= nmax(2)+1)then
          Parrivals = P(1)*P(2)
          Pinternal = Pint(ch,int)
PSum = PSum + ((1-Pbd)*Prep*Parrivals*Pinternal
               *Z*Vopt(n(1),n(2),3,e)
             + Pbd*Prep*Parrivals*Pinternal
               *Z*Vopt(n(1),n(2),2,e)&
             + (1-Pbd)*(1-Prep)*Parrivals*Pinternal
               *Z*Vopt(n(1),n(2),2,e)&
             + (1-Prep)*Pbd*Parrivals*Pinternal
               *Z*Vopt(n(1),n(2),1,e))
          if(ch == choice) then
PSumind = PSumind + Pinternal*Parrivals*Pbd
                    *Z*Vind(n(1),n(2),1,e)&
```

```
                    + Pinternal*Parrivals*(1-Pbd)
                      *Z*Vind(n(1),n(2),2,e)
            end if
          end if
        end do
      end do
     rew = RMat(ch) + PSum
     if(rew > maxi) maxi = rew
     if(ch == choice) then
        Rewindex = RMat(choice) + PSumind
     end if
    end if
  end do

Vind(IS(1),IS(2),2,f) = Rewindex Vopt(IS(1),IS(2),2,f) = maxi
return end
!=============================================================
subroutine ZERO_up(IS,nmax,Prep,Z,Vopt,Vind,e,f)
implicit none
integer :: e,f
integer, dimension(2) :: IS,nmax
double precision :: Z,Prep
double precision, dimension(nmax(1)+1,nmax(2)+1,2,2) :: Vind
double precision, dimension(nmax(1)+1,nmax(2)+1,3,2) :: Vopt

Vopt(IS(1),IS(2),1,f) = Prep*Prep*Z*Vopt(IS(1),IS(2),3,e)
                      + 2*(Prep*(1-Prep)*Z*Vopt(IS(1),IS(2),2,e))
                      + (1-Prep)*(1-Prep)*Z*Vopt(IS(1),IS(2),1,e)

Vind(IS(1),IS(2),1,f) = Prep*Z*Vind(IS(1),IS(2),2,e)
                      + (1-Prep)*Z*Vind(IS(1),IS(2),1,e)
return end
!=============================================================
subroutine GET_RMAX_RMIN(c,nmax,f,Vind,Rmax,Rmin)
implicit none
integer :: M,m1,m2,f
integer, dimension(2) :: c,nmax
double precision :: rew,mini,maxi
double precision,dimension(3) :: Rmax,Rmin
double precision, dimension(nmax(1)+1,nmax(2)+1,2,2) :: Vind

Rmax = 0.0d0
Rmin = 0.0d0

do M = 1,3
  rew = 0.0d0
  if(M == 1) then
    maxi = -10000.0d0
    mini = 10000.0d0
    do m1 = 1,c(1)
      do m2 = 1,c(2)
        if(m1 == 1 .and. m2 == 1) then
        else if(m1 == c(1) .and. m2 == c(2)) then
        else
rew = Vind(m1,m2,1,f) + Vind(c(1)-m1+1,c(2)-m2+1,1,f)
        if(rew > maxi) maxi = rew
        if(rew < mini) mini = rew
        end if
      end do
    end do
```

```
    Rmax(1) = maxi
    Rmin(1) = mini
  else if(M == 2) then
    maxi = -10000.0d0
    mini = 10000.0d0
    do m1 = 1,c(1)
      do m2 = 1,c(2)
        if(m1 == 1 .and. m2 ==1)then
        else if(m1 == c(1) .and. m2 == c(2))then
        else
rew = Vind(m1,m2,1,f) + Vind(c(1)-m1+1,c(2)-m2+1,2,f)
        if(rew > maxi) maxi = rew
        if(rew < mini) mini = rew
        end if
      end do
    end do
    Rmax(2) = maxi
    Rmin(2) = mini
  else if (M == 3) then
    maxi = -10000.0d0
    mini = 10000.0d0
    do m1 = 1,c(1)
      do m2 = 1,c(2)
        if(m1 == 1 .and. m2 ==1)then
        else if(m1 /= c(1) .and. m2 /= c(2))then
        else
rew = Vind(m1,m2,2,f) + Vind(c(1)-m1+1,c(2)-m2+1,2,f)
        if(rew > maxi) maxi = rew
        if(rew < mini) mini = rew
        end if
      end do
    end do
    Rmax(3) = maxi
    Rmin(3) = mini
  end if
end do return end
!=============================================================
subroutine GETsubopt(y,SIM,BIG,SMALL,Ropt,Rmax,Rmin,Subopt)
implicit none
integer :: i,y,h,SIM
double precision, dimension(3) :: Ropt,Rmax,Rmin
double precision, dimension(2,3) :: SMALL,BIG
double precision, dimension(2,3,SIM) :: Subopt

do h = 1,3
  Subopt(1,h,y) = Ropt(h) - Rmax(h)
  Subopt(2,h,y) = Ropt(h) - Rmin(h)
end do

do i = 1,2
  do h = 1,3
    if(Subopt(i,h,y) >= BIG(i,h)) then
      BIG(i,h) = Subopt(i,h,y)
    end if
    if(Subopt(i,h,y) <= SMALL(i,h)) then
      SMALL(i,h) = Subopt(i,h,y)
    end if
  end do
end do
return
```

```
end
!=============================================================
subroutine GETstats(SIM,BIG,SMALL,Subopt,RoptMEAN)
implicit none
integer :: SIM,h,i,j,k,l
double precision :: t
double precision, dimension(3) :: RoptMEAN
double precision, dimension(2,3) :: SMALL,BIG,MEAN,MEDIAN,STD,sum,sum2,PERCENT
double precision, dimension(2,3,SIM) :: Subopt,ORDER

ORDER = 1.0e+20
do l = 1,2
  do h = 1,3
    do j = 1,SIM
      do i = 1,SIM
        if(ORDER(l,h,j) >= Subopt(l,h,i)) then
          ORDER(l,h,j) = Subopt(l,h,i)
          k = i
        end if
      end do
      Subopt(l,h,k) = 1.0e+21
    end do
    j = 0
    do i = 1,SIM
      if(ORDER(l,h,i) > 1.0e-8) j = j + 1
    end do
    if (mod(j,2) == 1 .and. j > 0)then
      MEDIAN(l,h) = ORDER(l,h,SIM - j + ((j+1)/2))
    else if (mod(j,2) == 0 .and. j > 0)then
      MEDIAN(l,h) = (ORDER(l,h,SIM - j + (j/2))
                  + ORDER(l,h,SIM - j + ((j/2)+1)))/2d0
    else if (j == 0) then
      MEDIAN(l,h) = 0d0
    end if

    sum(l,h) = 0d0
    sum2(l,h) = 0d0
    do i = 1,SIM
      sum(l,h) = sum(l,h) + ORDER(l,h,i)
      sum2(l,h) = sum2(l,h) + (ORDER(l,h,i))**2
    end do

    MEAN(l,h) = sum(l,h)/SIM*1d0
    t = (((1d0/SIM*1d0)*sum2(l,h))) - (MEAN(l,h)**2)
    if(t > 0)then
      STD(l,h) = sqrt((((1d0/SIM*1d0)*sum2(l,h))) - (MEAN(l,h)**2))
    else
      STD(l,h) = 0d0
    end if
    PERCENT(l,h) = 100.0d0 -(100d0*j/SIM*1d0)
  end do
end do

write(unit=6,fmt="(a)")"============================================="
write(unit=6,fmt="(a)")" SUMMARY STATISTICS "
write(unit=6,fmt="(a,f12.6,a,f12.6,a,f12.6,a,f12.6,a,f12.6,a,f12.6)")&
&"MIN",SMALL(1,1),"",SMALL(1,2),"",SMALL(1,3),"
MIN",SMALL(2,1),"",SMALL(2,2),"",SMALL(2,3)
write(unit=6,fmt="(a,f12.6,a,f12.6,a,f12.6,a,f12.6,a,f12.6,a,f12.6)")&
&"MED",MEDIAN(1,1),"",MEDIAN(1,2),"",MEDIAN(1,3),"
```

```
MED",MEDIAN(2,1),"",MEDIAN(2,2),"",MEDIAN(2,3)
write(unit=6,fmt="(a,f12.6,a,f12.6,a,f12.6,a,f12.6,a,f12.6,a,f12.6)")&
&"MAX",BIG(1,1),"",BIG(1,2),"",BIG(1,3),"
MAX",BIG(2,1),"",BIG(2,2),"",BIG(2,3) write(unit=6,fmt="(a)")""
write(unit=6,fmt="(a,f11.6,a,f12.6,a,f12.6,a,f11.6,a,f12.6,a,f12.6)")&
&"MEAN",MEAN(1,1),"",MEAN(1,2),"",MEAN(1,3),"
MEAN",MEAN(2,1),"",MEAN(2,2),"",MEAN(2,3)
write(unit=6,fmt="(a,f12.6,a,f12.6,a,f12.6,a,f12.6,a,f12.6,a,f12.6)")&
&"STD",STD(1,1),"",STD(1,2),"",STD(1,3),"
STD",STD(2,1),"",STD(2,2),"",STD(2,3) write(unit=6,fmt="(a)")""
write(unit=6,fmt="(a,f12.6,a,f12.6,a,f12.6,a,f11.6,a,f12.6,a,f12.6)")&
&"RAV",RoptMEAN(1),"",RoptMEAN(2),"",RoptMEAN(3),"
RAV",RoptMEAN(1),"",RoptMEAN(2),"",RoptMEAN(3)
write(unit=6,fmt="(a)")""
write(unit=6,fmt="(a,f12.6,a,f12.6,a,f12.6,a,f11.6,a,f12.6,a,f12.6)")&
&"QUO",MEAN(1,1)/RoptMEAN(1),"",MEAN(1,2)/RoptMEAN(2),"",MEAN(1,3)/RoptMEAN(3),&
&"
QUO",MEAN(2,1)/RoptMEAN(1),"",MEAN(2,2)/RoptMEAN(2),"",MEAN(2,3)/RoptMEAN(3)
write(unit=6,fmt="(a)")"================================================="
write(unit=6,fmt="(a)")" " write(unit=6,fmt="(a)")" "
write(unit=6,fmt="(a)")" "

write(unit=7,fmt="(a)")"================================================="
write(unit=7,fmt="(a)")" SUMMARY STATISTICS "
write(unit=7,fmt="(a,f12.6,a,f12.6,a,f12.6,a,f12.6,a,f12.6,a,f12.6)")&
&"MIN",SMALL(1,1),"",SMALL(1,2),"",SMALL(1,3),"
MIN",SMALL(2,1),"",SMALL(2,2),"",SMALL(2,3)
write(unit=7,fmt="(a,f12.6,a,f12.6,a,f12.6,a,f12.6,a,f12.6,a,f12.6)")&
&"MED",MEDIAN(1,1),"",MEDIAN(1,2),"",MEDIAN(1,3),"
MED",MEDIAN(2,1),"",MEDIAN(2,2),"",MEDIAN(2,3)
write(unit=7,fmt="(a,f12.6,a,f12.6,a,f12.6,a,f12.6,a,f12.6,a,f12.6)")&
&"MAX",BIG(1,1),"",BIG(1,2),"",BIG(1,3),"
MAX",BIG(2,1),"",BIG(2,2),"",BIG(2,3) write(unit=7,fmt="(a)")""
write(unit=7,fmt="(a,f11.6,a,f12.6,a,f12.6,a,f11.6,a,f12.6,a,f12.6)")&
&"MEAN",MEAN(1,1),"",MEAN(1,2),"",MEAN(1,3),"
MEAN",MEAN(2,1),"",MEAN(2,2),"",MEAN(2,3)
write(unit=7,fmt="(a,f12.6,a,f12.6,a,f12.6,a,f12.6,a,f12.6,a,f12.6)")&
&"STD",STD(1,1),"",STD(1,2),"",STD(1,3),"
STD",STD(2,1),"",STD(2,2),"",STD(2,3) write(unit=7,fmt="(a)")""
write(unit=7,fmt="(a,f12.6,a,f12.6,a,f12.6,a,f11.6,a,f12.6,a,f12.6)")&
&"R.AV",RoptMEAN(1),"",RoptMEAN(2),"",RoptMEAN(3),"
R.AV",RoptMEAN(1),"",RoptMEAN(2),"",RoptMEAN(3)
write(unit=7,fmt="(a)")""
write(unit=7,fmt="(a,f12.6,a,f12.6,a,f12.6,a,f11.6,a,f12.6,a,f12.6)")&
&"QUO",MEAN(1,1)/RoptMEAN(1),"",MEAN(1,2)/RoptMEAN(2),"",MEAN(1,3)/RoptMEAN(3),&
&"
R.AV",MEAN(2,1)/RoptMEAN(1),"",MEAN(2,2)/RoptMEAN(2),"",MEAN(2,3)/RoptMEAN(3)
write(unit=7,fmt="(a)")"================================================="
write(unit=7,fmt="(a)")" " write(unit=7,fmt="(a)")" "
write(unit=7,fmt="(a)")" "

return end
!============================================================
subroutine GETrewards
(SIM,nmax,M,c,Rmat,Parr,Pint,Prep,Pbd,INDEX,alpha,Z,Ropt,Rmax,Rmin)
implicit none
integer :: M,n1,n2,e,f,q,choice,SIM
integer, dimension(2) :: c,IS,nmax
integer, dimension(3) :: choicevec
integer, dimension(3,3) :: choicemat
```

```
double precision :: maxi,Pbd,Prep,Z,alpha
double precision, dimension(2) :: INDEX,Parr
double precision, dimension(3) :: RMat,Ropt,Rmax,Rmin
double precision, dimension(3,3) :: Pint
double precision, dimension(nmax(1)+1,nmax(2)+1,2,2) :: Vind
double precision, dimension(nmax(1)+1,nmax(2)+1,3,2) :: Vopt

Vind = 0d0
Vopt = 0d0
Rmax = 0d0
Rmin = 0d0
Ropt = 0d0

do q = 1,int((-1.0d0/alpha)*log((alpha*10e-8)/(2*5.0d0)))+1
  if (mod(q,2) == 1) then
    e = 1
    f = 2
  else if (mod(q,2) == 0) then
    e = 2
    f = 1
  end if

  do n1 = 1,nmax(1)+1
    do n2 = 1,nmax(2)+1
      IS = (/n1,n2/)
        call GETchoicemat(IS,choicemat)
        call GETchoicevec(IS,choicevec)
        maxi = -10000.0d0
        choice = 0
        do i = 1,2
          if(INDEX(i) > maxi .and. IS(i) > 1) then
            choice = i
            maxi = INDEX(i)
          end if
        end do
        if(IS(1)==1 .and. IS(2)==1) then
          choice = 3
        end if
call ZERO_up(IS,nmax,Prep,Z,Vopt,Vind,e,f) call
ONE_up(IS,nmax,choice,choicevec,Parr,Pint,Pbd,Prep,RMat,Z,Vopt,Vind,e,f)
call TWO_up(IS,nmax,choicemat,Parr,Pint,Pbd,RMat,Z,Vopt,e,f)
    end do
  end do
end do

Ropt(1) = Vopt(c(1),c(2),1,f)
Ropt(2) = Vopt(c(1),c(2),2,f)
Ropt(3) = Vopt(c(1),c(2),3,f)

call GET_RMAX_RMIN(c,nmax,f,Vind,Rmax,Rmin)

return end
!============================================================
```

Appendix I

```
program stochastic_MAB

implicit none
integer :: i,j,k,l,h,d,b,type,count,countcomp,E,SIZE
integer :: RANDOM,Machine,y,cond,c,bandit1,SIM
integer, dimension(3) :: seed
integer, dimension(4) :: argmax,STATE
double precision :: SMALL,BIG,MEAN,MEDIAN,STD,t,alpha,Z,sum,sum2,PERCENT
double precision :: Ropt,Rind
double precision, dimension(4) :: r,G,IND
double precision, dimension(3,3) :: Pmach
double precision, dimension(4,4) :: P,Rmat integer, dimension(15,4) :: S
double precision, dimension(15,4) :: A,AV
double precision, dimension(4,4) :: INDEX
double precision, dimension(4,4,4) :: Pmat
double precision, dimension(500) :: Subopt,ORDER

type = 7
SIM = 500
seed = (/21689,9112,14519/)
E = 4

do j = 15,1,-1
  d = j
  do i = 3,0,-1
    t = d/2**i
    if (t < 1) S(j,(4-i)) = 0
    if (t>= 1) then
      S(j,(4-i)) = 1
      d = mod(d,2**i)
    end if
  end do
end do

do c = 1,7
  if (c == 1) then
    alpha = 0.1d0
  else if (c == 2) then
    alpha = 0.05d0
  else if (c == 3) then
    alpha = 0.025d0
  else if (c == 4) then
    alpha = 0.01d0
  else if (c == 5) then
    alpha = 0.005d0
  else if (c == 6) then
    alpha = 0.0025d0
  else if (c == 7) then
```

```
      alpha = 0.001d0
    end if

    Subopt = 0d0
    order = 0d0
    SMALL = 1.0e+20
    BIG = -1*(1.0e+20)

    do y = 1,SIM

      Ropt = 0d0
      Rind = 0d0

      call GETrandom(seed,RANDOM)
      call GETmachine(seed,Machine)
      Machine = Machine + 1

      h = 0
      do i = 1,4
        do j = 1,4
          do k = 1,4
            do l = 1,4
              h = h + 1
              if (h == RANDOM) then
                STATE = (/i,j,k,l/)
              end if
            end do
          end do
        end do
      end do

      call GETPmatrix(seed,Pmat)
      call GETRmatrix(seed,Rmat)
      call GETMmatrix(seed,Pmach)
!     call GETMmatrix_non_decreasing(seed,Pmach)

      Z = exp(-1*(alpha))
      do b = 1,4
        do i = 1,4
          do j = 1,4
            P(i,j) = Pmat(b,i,j)
          end do
          r(i) = Rmat(b,i)
        end do
        h = 15
        G = 0
        IND = 0
        argmax = 0
        A = 0
        A(15,:) = (/1,1,1,1/)
        AV = 0
        do k = 1,E
          if (k > 1) call GETsubset(E,argmax,k,h)
            count = 0
            do i = 1,E
              count = count + S(h,i)
              countcomp = E - count
            end do
            if (k > 1) call GETAmatrix(AV,S,count,countcomp,P,Z,h,A)
            call GETindices(b,k,S,h,r,A,argmax,G,IND,INDEX)
```

```
    end do
  end do

  call Getbandit1(INDEX,bandit1,cond)
  call GETrewards(Machine,STATE,Rmat,Pmat,Pmach,INDEX,alpha,bandit1,Ropt,Rind)

  Subopt(y) = Ropt - Rind

  if(Subopt(y) >= BIG) then
    BIG = Subopt(y)
  end if
  if(Subopt(y) <= SMALL) then
    SMALL = Subopt(y)
  end if

end do

ORDER = 1.0e+20

do j = 1,SIM
  do i = 1,SIM
    if(ORDER(j) >= Subopt(i)) then
      ORDER(j) = Subopt(i)
      k = i
    end if
  end do
  Subopt(k) = 1.0e+21
end do

j = 0
do i = 1,SIM
  if(ORDER(i) > 1.0e-8) j = j + 1
end do
if (mod(j,2) == 1 .and. j > 0)then
  MEDIAN = ORDER(SIM - j + ((j+1)/2))
else if (mod(j,2) == 0 .and. j > 0)then
  MEDIAN = (ORDER(SIM - j + (j/2)) + ORDER(SIM - j + ((j/2)+1)))/2d0
else if (j == 0) then
  MEDIAN = 0d0
end if

sum = 0d0
sum2 = 0d0
do i = 1,SIM
  sum = sum + ORDER(i)
  sum2 = sum2 + (ORDER(i))**2
end do

MEAN = sum/SIM*1d0
t = (((1d0/SIM*1d0)*sum2)) - (MEAN**2)
if(t > 0)then
  STD = sqrt((((1d0/SIM*1d0)*sum2)) - (MEAN**2))
else
  STD = 0d0
end if
PERCENT = 100.0d0 -(100d0*j/SIM*1d0)

write(unit = 6, fmt= "(a)")"========================================="
write(unit = 6, fmt= "(a,f6.4)") "alpha =    " ,alpha
write(unit = 6, fmt= "(a,f12.6,a,f12.6,a,f12.6)")
```

```
  "(1) Minimum", SMALL,"     (2) Median ",MEDIAN,"     (3) Maximum",BIG
  write(unit = 6, fmt= "(a,f12.6,a,f12.6,a,f12.6)")
  "(4) Mean   ", MEAN,"     (5) Std Dev",STD,"     (6) Percent",PERCENT
  write(unit = 6, fmt= "(a)")"========================================="

  open(unit = 7, file = "STOCHASTIC.dat")
!  open(unit = 7, file = "STOCHASTIC_non_decreasing.dat")

  write(unit = type, fmt= "(a)")"========================================="
  write(unit = type, fmt= "(a,f6.4)") "alpha =   " ,alpha
  write(unit = type, fmt= "(a,f12.6,a,f12.6,a,f12.6)")
  "(1) Minimum", SMALL,"     (2) Median ",MEDIAN,"    (3) Maximum",BIG
  write(unit = type, fmt= "(a,f12.6,a,f12.6,a,f12.6)")
  "(4) Mean   ", MEAN,"     (5) Std Dev",STD,"    (6) Percent",PERCENT
  write(unit = type, fmt= "(a)")"========================================="
end do

end program
!=====================================================================
subroutine GETsubset(E,argmax,k,h)
implicit none
integer :: k,h,E
integer, dimension(4) :: argmax

h = h - 2**(E - argmax(k - 1))
return
end
!=====================================================================
subroutine GETAmatrix(AV,S,count,countcomp,P,Z,h,A)
implicit none
integer :: d,g,i,j,h,f,k,count,countcomp,E
double precision :: matsum,Z,sum
integer, dimension(15,4) :: S
double precision, dimension(4,4) :: P
double precision, dimension(15,4) :: A,AV
double precision, dimension(countcomp,countcomp) :: Mmat,Pmat
double precision, dimension(countcomp,1) :: Pvec,Psum,Product
double precision, dimension(count) :: V1,V2

E = 4
!-----------------------------------------------
Mmat = 0d0
!-----------------------------------------------
d = 0
do i = 1,E
  if(S(h,i) == 0) then
    if(d < countcomp) d = d + 1
    Mmat(d,d) = 1 - Z*P(i,i)
    g = d
    do j = 1,E
      if (i < j) then
        if (S(h,j) == 0) then
          if (g < countcomp) g = g + 1
          Mmat(d,g) = -Z*P(i,j)
          Mmat(g,d) = -Z*P(j,i)
        end if
      end if
    end do
  end if
end do
```

```
!-------------------------------------------------
Psum = 0d0
Pvec = 0d0
Pmat = 0d0
d = 0
do i = 1,E
  g = 0
  if (S(h,i) == 0) then
    if (d < countcomp) d = d + 1
    do j = 1,E
      if (S(h,j) == 0) then
        if(g < countcomp) g = g + 1
        Pmat(d,g) = P(i,j)
      end if
    end do
  end if
end do
!-------------------------------------------------
do i = 1,countcomp
  do j = 1,countcomp
    Psum(i,1) = Psum(i,1) + Pmat(i,j)
  end do
end do
do i = 1,countcomp
  Pvec(i,1) = (1 - Psum(i,1))*Z
end do
!-------------------------------------------------
call GETinverse(countcomp,Mmat)
Product = 0d0
do i = 1,countcomp
  sum = 0d0
  do j = 1,countcomp
    sum = sum + (Mmat(i,j)*Pvec(j,1))
  end do
  Product(i,1) = sum
end do
Pvec = Product
!-------------------------------------------------
f = 0
do k = 1,4
  if (S(h,k) == 0) then
    if (f < countcomp) f = f + 1
    AV(h,k) = Pvec(f,1)
  end if
end do

d = 0
do i = 1,E
  matsum = 0
  if (S(h,i) == 1) then
    if (d < count) d = d + 1
    do j = 1,E
      if (S(h,j) == 0) matsum = matsum + (P(i,j) * AV(h,j))
    end do
    V1(d) = Z*matsum
    matsum = 0
    do j = 1,E
      if (S(h,j) == 1) matsum = matsum + (P(i,j) * Z)
    end do
    V2(d) = matsum
```

```
    AV(h,i) = V1(d) + V2(d)
  end if
end do

AV(15,:) = (/Z,Z,Z,Z/)

f = 0
do k = 1,4
  if (S(h,k) == 1) then
    if (f < count) f = f + 1
    A(h,k) = (1 - AV(h,k))/(1 - Z)
  end if
end do
return
end
!=====================================================================
subroutine GETindices(b,k,S,h,r,A,argmax,G,IND,INDEX)
implicit none
integer :: b,i,h,k,l,j,E
double precision :: max,sum
integer, dimension(4) :: argmax
double precision, dimension(4) :: r,G,IND
double precision, dimension(15,4) :: A
integer, dimension(15,4) :: S
double precision, dimension(4,4) :: INDEX

if (k == 1) then
  max = -1000000
  do i = 1,4
    if (r(i)/A(h,i) > max) then
      max = r(i)/A(h,i)
      argmax(1) = i
    end if
  end do
  G(argmax(1)) = max
  IND(argmax(1)) = G(argmax(1))
else
  E = 4
  max = -1000000
  do i = 1,4
    l = 15
    sum = 0
    if (S(h,i) == 1) then
      do j = 1,k-1
        sum = sum + A(l,i)*G(argmax(j))
        l = l - 2**(E - argmax(j))
      end do
      if ((r(i) - sum)/A(h,i) > max) then
        max = (r(i) - sum)/A(h,i)
        argmax(k) = i
      end if
    end if
  end do
  G(argmax(k)) = max
  IND(argmax(k)) = G(argmax(k)) + IND(argmax(k - 1))
end if

INDEX(b,argmax(k)) = IND(argmax(k))
return
end
```

```
!=============================================================
subroutine GETinverse(countcomp,Mmat)
implicit none
integer :: i,j,countcomp
double precision :: a,b,c,d,determinant
double precision, dimension(countcomp,countcomp) :: Mmat
double precision, dimension(3,3) :: Cofactor

if (countcomp == 1) then
  Mmat(1,1) = 1d0/Mmat(1,1)
else if (countcomp == 2) then
  a = Mmat(1,1)
  b = Mmat(1,2)
  c = Mmat(2,1)
  d = Mmat(2,2)
  determinant = (a*d) - (b*c)
  Mmat(1,1) = d/determinant
  Mmat(1,2) = -1d0*(b/determinant)
  Mmat(2,1) = -1d0*(c/determinant)
  Mmat(2,2) = a/determinant
else if (countcomp == 3) then
  Cofactor(1,1) = ((Mmat(2,2)*Mmat(3,3)) - (Mmat(2,3)*Mmat(3,2)))
  Cofactor(1,2) = -1d0*((Mmat(2,1)*Mmat(3,3)) - (Mmat(2,3)*Mmat(3,1)))
  Cofactor(1,3) = ((Mmat(2,1)*Mmat(3,2)) - (Mmat(2,2)*Mmat(3,1)))

  Cofactor(2,1) = -1d0*((Mmat(1,2)*Mmat(3,3)) - (Mmat(1,3)*Mmat(3,2)))
  Cofactor(2,2) = ((Mmat(1,1)*Mmat(3,3)) - (Mmat(1,3)*Mmat(3,1)))
  Cofactor(2,3) = -1d0*((Mmat(1,1)*Mmat(3,2)) - (Mmat(1,2)*Mmat(3,1)))

  Cofactor(3,1) = ((Mmat(1,2)*Mmat(2,3)) - (Mmat(1,3)*Mmat(2,2)))
  Cofactor(3,2) = -1d0*((Mmat(1,1)*Mmat(2,3)) - (Mmat(1,3)*Mmat(2,1)))
  Cofactor(3,3) = ((Mmat(1,1)*Mmat(2,2)) - (Mmat(1,2)*Mmat(2,1)))

  determinant = Mmat(1,1)*Cofactor(1,1) + Mmat(1,2)*Cofactor(1,2)
              + Mmat(1,3)*Cofactor(1,3)

  do i = 1,3
    do j = 1,3
      Mmat(i,j) = Cofactor(j,i)/determinant
    end do
  end do
end if
return
end
!=============================================================
subroutine GETPmatrix(seed,Pmat) implicit none
integer :: i,j,k
integer, dimension(3) :: seed
double precision :: UNIFORM,sum,L,U
double precision, dimension(4,4,4) :: PMat

L = 0.1
U = 0.9

do k = 1,4
  do i = 1,4
    do j = 1,4
      call GETuniform(seed,L,U,UNIFORM)
      PMat(k,i,j) = UNIFORM
    end do
```

```
  end do
end do

do k = 1,4
  do i = 1,4
    sum = 0d0
    do j = 1,4
      sum = sum + PMat(k,i,j)
    end do
    do j = 1,4
      PMat(k,i,j) = Pmat(k,i,j)*(1.0d0/sum)
    end do
  end do
end do
return
end
!=============================================================
subroutine GETRmatrix(seed,Rmat)
implicit none
integer :: i,j
integer, dimension(3) :: seed
double precision :: L,U,x
double precision, dimension(4,4) :: RMat

L = 2d0
U = 5d0

do i = 1,4
  do j = 1,4
    call GETuniform(seed,L,U,x)
    RMat(i,j) = x
  end do
end do
return
end
!=============================================================
subroutine GETMmatrix(seed,Pmach)
implicit none
integer :: i,j
integer, dimension(3) :: seed
double precision :: L,U,x,sum
double precision, dimension(3,3) :: Pmach

L = 0.1d0
U = 0.9d0

do i = 1,3
  do j = 1,3
    call GETuniform(seed,L,U,x)
    Pmach(i,j) = x
  end do
end do

do i = 1,3
  sum = 0d0
  do j = 1,3
    sum = sum + Pmach(i,j)
  end do
  do j = 1,3
    Pmach(i,j) = Pmach(i,j)*(1.0d0/sum)
```

```
  end do
end do
return
end
!=============================================================
subroutine GETMmatrix_non_decreasing(seed,Pmach)
implicit none
integer :: i,j
integer, dimension(3) :: seed
double precision :: L,U,x,sum
double precision, dimension(3,3) :: Pmach

L = 0.1d0
U = 0.9d0

do i = 1,3
  do j = 1,3
    if(j < i) then
      Pmach(i,j) = 0.0d0
    else if (j >= i) then
      call GETuniform(seed,L,U,x)
      Pmach(i,j) = x
    end if
  end do
end do

do i = 1,3
  sum = 0d0
  do j = 1,3
    sum = sum + Pmach(i,j)
  end do
  do j = 1,3
    Pmach(i,j) = Pmach(i,j)*(1.0d0/sum)
  end do
end do return end
!=============================================================
subroutine GETuniform(seed,L,U,x)
implicit none
integer, dimension(3) :: seed
double precision :: r,s,x,L,U

seed(1) = mod(171*seed(1),30269)
seed(2) = mod(172*seed(2),30307)
seed(3) = mod(170*seed(3),30323)

s = seed(1)*1d0/30269 + seed(2)*1d0/30307 + seed(3)*1d0/30323
r = s - int(s)

x = L + (U-L)*r
return
end
!=============================================================
subroutine GETrandom(seed,x)
implicit none
integer :: x
integer, dimension(3) :: seed
double precision :: r,s

seed(1) = mod(171*seed(1),30269)
seed(2) = mod(172*seed(2),30307)
```

```
seed(3) = mod(170*seed(3),30323)

s = seed(1)*1d0/30269 + seed(2)*1d0/30307 + seed(3)*1d0/30323
r = s - int(s)

x = int(256*r)
return
end
!=============================================================
subroutine GETmachine(seed,x)
implicit none
integer :: x
integer, dimension(3) :: seed
double precision :: r,s

seed(1) = mod(171*seed(1),30269)
seed(2) = mod(172*seed(2),30307)
seed(3) = mod(170*seed(3),30323)

s = seed(1)*1d0/30269 + seed(2)*1d0/30307 + seed(3)*1d0/30323
r = s - int(s)

x = int(3*r)
return
end
!=============================================================
subroutine GETbandit1(IND,bandit1,cond)
implicit none
integer :: i,j,c,cond,bandit1
double precision :: max double
precision, dimension(3) :: Gsub
double precision, dimension(4) :: min
double precision, dimension(4,4) :: IND

min = 0d0
do i= 1,4
  min(i) = 1000000d0
  do j = 1,4
    if (IND(i,j) < min(i)) then
      min(i) = IND(i,j)
    end if
  end do
end do

do i = 1,3
  max = -10000
  do j = 1,4
    if (min(j) > max) then
      max = min(j)
      Gsub(i) = min(j)
      c = j
    end if
  end do
  if (i == 1) then
    bandit1 = c
  end if
  min(c) = -100000
end do

if (Gsub(1) /= Gsub(2) .and. Gsub(2) /= Gsub(3) .and. Gsub(1) /= Gsub(3)) then
```

```
  cond = 1
else
  cond = 2
end if
return
end
!=============================================================
subroutine GETrewards(M,c,Rmat,Pmat,Pmach,INDEX,alpha,bandit1,Rewopt,Rewpol)
implicit none
integer :: M,q,e,f,x1,x2,x3,x4,mach,a,bandit1,b,bandit2,choice,choice1,choice2
integer, dimension(4) :: c,IS,x
double precision :: alpha,PSum,rew,maxi,Rewopt,Rewpol
double precision, dimension(3,3) :: Pmach
double precision, dimension(4,4) :: Rmat
double precision, dimension(4,4) :: INDEX
double precision, dimension(4,4,4) :: Pmat
double precision, dimension(4,4,4,4,3,2) :: Vopt,Vpol

Vpol = 0d0
Vopt = 0d0
Rewopt = 0d0
Rewpol = 0d0
do q = 1,int((-1.0d0/alpha)*log((alpha*10e-8)/(2*5.0d0)))+1
  if (mod(q,2) == 1) then
    e = 1
    f = 2
  else if (mod(q,2) == 0) then
    e = 2
    f = 1
  end if
!-------------------------------------------------------------
  do x1 = 1,4
    do x2 = 1,4
      do x3 = 1,4
        do x4 = 1,4
          IS = (/x1,x2,x3,x4/)
!--------------------------POLICY-----------------------------
          PSum = 0d0
          do mach = 1,3
            PSum = PSum + Pmach(1,mach)*Vpol(x1,x2,x3,x4,mach,e)
          end do
          Vpol(x1,x2,x3,x4,1,f) = exp(-1.0d0*alpha)*PSum
!-------------------------------------------------------------
          PSum = 0d0
          x = IS
          do mach = 1,3
            do a = 1,4
              x(bandit1) = a
              PSum = PSum + Pmach(2,mach)*PMat(bandit1,IS(bandit1),a)&
              &*Vpol(x(1),x(2),x(3),x(4),mach,e)
            end do
          end do
Vpol(x1,x2,x3,x4,2,f) = RMat(bandit1,IS(bandit1)) + exp(-1.0d0*alpha)*PSum
!-------------------------------------------------------------
          call GETbandit2(INDEX,x1,x2,x3,x4,bandit1,bandit2)
          PSum = 0d0
          x = IS
          do mach = 1,3
            do a = 1,4
              x(bandit1) = a
```

```
                do b = 1,4
                  x(bandit2) = b
                  PSum = PSum + Pmach(3,mach)
                  *PMat(bandit1,IS(bandit1),a)
                  *PMat(bandit2,IS(bandit2),b)
                  *Vpol(x(1),x(2),x(3),x(4),mach,e)
                end do
              end do
            end do
Vpol(x1,x2,x3,x4,3,f) = RMat(bandit1,IS(bandit1)) + RMat(bandit2,IS(bandit2))&
          & + exp(-1.0d0*alpha)*PSum
!---------------------------OPTIMAL-------------------------
          PSum = 0d0
          do mach = 1,3
            PSum = PSum + Pmach(1,mach)*Vopt(x1,x2,x3,x4,mach,e)
          end do
          Vopt(x1,x2,x3,x4,1,f) = exp(-1.0d0*alpha)*PSum
!-----------------------------------------------------------
          rew = 0d0
          maxi = -10000.0d0
          do choice = 1,4
            PSum = 0d0
            x = IS
            do mach = 1,3
              do a = 1,4
                x(choice) = a
                PSum = PSum + Pmach(2,mach)
                *PMat(choice,IS(choice),a)
                *Vopt(x(1),x(2),x(3),x(4),mach,e)
              end do
            end do
            rew = RMat(choice,IS(choice)) + exp(-1.0d0*alpha)*PSum
            if(rew > maxi) then
              maxi = rew
            end if
          end do
          Vopt(x1,x2,x3,x4,2,f) = maxi
!-----------------------------------------------------------
          rew = 0d0
          maxi = -10000.0d0
          do choice1 = 1,3
            do choice2 = 1,4
              if(choice1 < choice2)then
                PSum = 0d0
                x = IS
                do mach = 1,3
                  do a = 1,4
                    x(choice1) = a
                    do b = 1,4
                      x(choice2) = b
                      PSum = PSum + Pmach(3,mach)*PMat(choice1,IS(choice1),a)
                      *PMat(choice2,IS(choice2),b)
                      *Vopt(x(1),x(2),x(3),x(4),mach,e)
                    end do
                  end do
                end do
                rew = RMat(choice1,IS(choice1)) + RMat(choice2,IS(choice2))&
                & + exp(-1.0d0*alpha)*PSum
                if(rew > maxi) then
                  maxi = rew
```

```
                end if
              end if
            end do
          end do
          Vopt(x1,x2,x3,x4,3,f) = maxi
        end do
      end do
    end do
  end do
end do

Rewopt = Vopt(c(1),c(2),c(3),c(4),M,f)
Rewpol = Vpol(c(1),c(2),c(3),c(4),M,f)
return
end
!=============================================================================
subroutine GETbandit2(INDEX,x1,x2,x3,x4,bandit1,bandit2)
implicit none
integer :: i,j,k,x1,x2,x3,x4,bandit1,bandit2
integer, dimension(3) :: b,x
integer, dimension(4) :: IS
double precision, dimension(4,4) :: INDEX

IS = (/x1,x2,x3,x4/)
j = 0
do i = 1,4
  if(i /= bandit1) then
    j = j + 1
    x(j) = IS(i)
    b(j) = i
  end if
end do

bandit2 = 0
do i = 1,3
  j = mod((i+1),3)
  if (j == 0) j = 3
  k = mod((i+2),3)
  if (k == 0) k = 3
    if((INDEX(b(i),x(i)) >= INDEX(b(j),x(j)))
      .and.(INDEX(b(i),x(i)) >= INDEX(b(k),x(k))) ) then
      bandit2 = b(i)
    end if
end do
return
end
!=============================================================================
```

Adv. Appl. Prob. **33,** 365–390 (2001)
Printed in Northern Ireland

THE PERFORMANCE OF INDEX-BASED POLICIES FOR BANDIT PROBLEMS WITH STOCHASTIC MACHINE AVAILABILITY

R. T. DUNN * AND
K. D. GLAZEBROOK,* ** *University of Newcastle upon Tyne*

Abstract

We consider generalisations of two classical stochastic scheduling models, namely the discounted branching bandit and the discounted multi-armed bandit, to the case where the collection of machines available for processing is itself a stochastic process. Under rather mild conditions on the machine availability process we obtain performance guarantees for a range of controls based on Gittins indices. Various forms of asymptotic optimality are established for index-based limit policies as the discount rate approaches 0.

Keywords: Average-overtaking optimal; average-reward optimal; branching bandit; discounted rewards; Gittins index; machine breakdowns; multi-armed bandit problem; parallel machines; suboptimality bound

AMS 2000 Subject Classification: Primary 90B36
Secondary 68M20; 90B22

1. Introduction

Since the work of Gittins and Jones (1974), who established the optimality of Gittins index policies for multi-armed bandits with discounted rewards earned over an infinite horizon, a range of extensions to this classical model have been discussed in the literature. Several of these developments are related to the current paper. Firstly, models incorporating a branching mechanism for processed projects have been used to model open systems in which new jobs arrive for processing and also scheduling models with precedence constraints. See, for example, Glazebrook (1976) and Whittle (1981). Weiss (1988) followed by Bertsimas and Niño-Mora (1996) established the optimality of index policies for general families of branching bandit models. Secondly, Glazebrook (1984), (1987) demonstrated that index policies remain optimal for a range of extensions of the classical model in which the single machine providing service is subject to breakdown and repair. Stochastic scheduling models incorporating machine breakdowns have been studied by many authors since. See, for example, Birge *et al.* (1990) and Pinedo and Rammouz (1988). Finally, considerable effort has been devoted to the analysis of models in which the single machine/server of the classical results is replaced by a collection of identical machines which are permanently available and which work in parallel. From this work it has become clear that parallel machine stochastic scheduling problems are in general much less tractable than their single machine counterparts. See, for example, Weber (1982), Weber *et al.* (1986) and Weiss (1990), (1992), (1995) for important contributions.

Significant progress with parallel machine problems has followed Bertsimas and Niño-Mora's (1996) ground-breaking discussion of Gittins indexation from the perspective of the

Received 26 October 2000; revision received 11 January 2001.
* Postal address: Department of Statistics, University of Newcastle upon Tyne, Newcastle upon Tyne, NE1 7RU, UK.
** Email address: kevin.glazebrook@newcastle.ac.uk

achievable region approach to stochastic optimisation. See Dacre *et al.* (1999) for an extended account of this approach. Elements of Bertsimas and Niño-Mora's (1996) analysis have been developed to provide information on the closeness-to-optimality of Gittins index-based heuristics for a range of stochastic scheduling models on parallel machines. Various forms of asymptotic optimality of these heuristics have been established. See, for example, Dacre *et al.* (1999), Glazebrook and Garbe (1999) and Glazebrook and Niño-Mora (2001). Most significantly for the present work, Glazebrook and Wilkinson (2000) analysed index-based policies for the classical multi-armed bandit on parallel machines. However, as indicated by the earlier work on breakdown models, intermittent or partial machine availability is a major practical consideration. The current paper's aim is to so develop the tools of analysis rooted in the achievable region approach and described, for example, in Glazebrook and Garbe (1999) and Glazebrook and Wilkinson (2000) in such a way as to facilitate analysis of heuristics for scheduling control in contexts where the set of machines available for processing evolves as a (reasonably general) stochastic process.

The scheduling models which concern us are described in Section 2. One (Model 1) is a discounted branching bandit while the other (Model 2) is a discounted multi-armed bandit. In both cases, service is provided by M machines which are identical in their processing capacity, but which are not always available for processing. Our formulation, in line with earlier work, uses discounting of the form $e^{-\alpha t}$, and hence total expected rewards from the implementation of any policy are $O(1/\alpha)$, where $\alpha > 0$ is a discount rate. Section 3 concerns our modelling of machine availability. Roughly speaking, we shall simply require of the machine availability process that the long-run proportion of epochs at which all M machines are available is positive. Examples demonstrate that the technical conditions concerned are satisfied by a range of standard models. In Section 4 we develop tools of analysis which are designed to give information on the degree of suboptimality of heuristic scheduling controls. These tools are utilised in Section 5 (Model 1) and Section 6 (Model 2) to evaluate the performance of Gittins index-based scheduling controls for our problems with intermittent machine availability. These simple index-based heuristics partition the jobs to be scheduled among the (potentially available) machines at time 0. Thereafter, the individual machines process according to index policies whenever they are up. In some cases the rewards from these simple controls come within $O(1)$ of optimality, while in others we can achieve $O(\alpha)$. *Inter alia*, we recover in Section 6 the $O(\alpha)$ result of Glazebrook and Wilkinson (2000) concerning multi-armed bandits on a fixed number of machines, as a special case. Much practical and theoretical interest attaches to problems with small discount rate α and to the related limit $\alpha \to 0$. In Section 7, we utilise the results of Sections 5 and 6 to describe forms of asymptotic optimality for limit policies derived from our Gittins index-based controls.

2. Scheduling models

We shall consider two models for the scheduling of stochastic jobs (sometimes called projects or bandits) on a collection of M machines, which are not always available for processing.

2.1. Model 1: discounted branching bandit

At each decision epoch $t \in \mathbb{N}$, M_t is a collection of identical machines which are available to process a collection of jobs. If $M < \infty$ is the maximum number of (available) machines, then $\{M_t, t \in \mathbb{N}\}$ is a stochastic process taking values in $2^{\{1,2,\ldots,M\}}$. Each of the jobs which require processing belongs to one of $C < \infty$ classes, labelled $\{1, 2, \ldots, C\}$. The *state* of the system at time $t \in \mathbb{N}$ is $\boldsymbol{N}(t) = \{N_1(t), \ldots, N_C(t)\}$ and records the number of jobs of

each class present and requiring service at t. At each $t \in \mathbb{N}$, jobs are chosen for processing, with (at most) one being allocated to each available machine. If a class i job is assigned to machine m at time t (which occurrence is registered by assigning the indicator function $\mathcal{X}_i^m(t)$ the value 1; it is 0 otherwise) then at time $t+1$ that class i job disappears to be replaced by $\boldsymbol{n}_i^{tm} \equiv \{n_{i1}^{tm}, n_{i2}^{tm}, \ldots, n_{iC}^{tm}\}$ jobs of classes $1, 2, \ldots, C$ respectively. These are the only changes to the system state between times t and $t+1$. We shall suppose that the vector $\boldsymbol{n}_i^{tm}$ has a decomposition

$$\boldsymbol{n}_i^{tm} = \boldsymbol{A}^{tm} + \tilde{\boldsymbol{n}}_i^{tm}. \tag{1}$$

In (1), $\boldsymbol{A}^{tm}$ is i-independent and is a collection of external arrivals to the system, while $\tilde{\boldsymbol{n}}_i^{tm}$ is a set of (internal) descendants of the serviced class i job. The latter enables us to model state transitions for jobs under processing. For a given i, the vectors $\boldsymbol{A}^{tm}$ and $\tilde{\boldsymbol{n}}_i^{tm}$ are i.i.d. as (t, m) varies. When $|M_t| > \sum_{j=1}^{C} N_j(t)$ there will certainly be idle machines at time t. An idle machine is deemed to be processing a class 0 job and we suppose that there are always M such jobs in the system, one for each machine. We extend the notation $\boldsymbol{n}_i^{tm}$ to include the case $i = 0$, but note that $n_{i0}^{tm} = 0$ for all choices of $i \neq 0$, t and m and $\boldsymbol{n}_0^{tm} = \{n_{01}^{tm}, n_{02}^{tm}, \ldots, n_{0C}^{tm}\} = \boldsymbol{A}^{tm}$.

From the above, the state evolution between t and $t+1$ is described by

$$\begin{aligned} N_i(t+1) &= N_i(t) + \sum_{m \in M_t} \sum_{j=0}^{C} \mathcal{X}_j^m(t)(n_{ji}^{tm} - \delta_{ij}), \qquad 1 \leq i \leq C, \\ N_0(t+1) &= N_0(t) = M. \end{aligned} \tag{2}$$

In (2), δ_{ij} is the Kronecker delta. Admissible controls (rules for choosing jobs for processing) are always assumed to be non-anticipative (based on the history of the process only). Through the paper, controls may need to meet other requirements. We shall always use $\mathcal{U}$ to denote the set of admissible controls with u a typical member.

The processing of a class i job on any machine at time t is assumed to earn a reward $r_i \int_t^{t+1} \mathrm{e}^{-\alpha s}\, \mathrm{d}s$ with $r_i > 0$, $1 \leq i \leq C$. The constant $\alpha > 0$ is a discount rate. No rewards are earned by the idle class 0. Rewards are additive across machines and over time. Write $E = \{0, 1, 2, \ldots, C\}$ for the full set of job classes. We shall express the total expected reward earned under control u from initial state $\boldsymbol{N}(0) = \boldsymbol{k}$ as

$$R^u(\boldsymbol{k}) = \sum_{j \in E} r_j x_j^u(\boldsymbol{k}), \tag{3}$$

where

$$x_j^u(\boldsymbol{k}) = \mathrm{E}_u \left\{ \int_0^\infty n_j(s)\, \mathrm{e}^{-\alpha s}\, \mathrm{d}s \;\middle|\; \boldsymbol{k} \right\}. \tag{4}$$

In (4), for each $t \in \mathbb{N}$, $n_j(s)$, $t \leq s < t+1$, is the number of class j jobs scheduled for processing at time t. Our goal is the analysis of the stochastic optimisation problem

$$R^{\mathrm{opt}}(\boldsymbol{k}) = \sup_{u \in \mathcal{U}} R^u(\boldsymbol{k}). \tag{5}$$

We shall guarantee the *stability* of this system under all admissible controls studied by requiring that the $|E| \times |E|$ matrix $\boldsymbol{I} - \boldsymbol{n}$ be positive definite. Here $\boldsymbol{I}$ is the identity and $\boldsymbol{n}$ has (i, j)th entry equal to $\mathrm{E}(n_{ij}^{tm})$, $i \in E$, $j \in E$. See Bertsimas and Niño-Mora (1996).

2.2. Model 2: discounted multi-armed bandit

There are $B \geq M$ bandits or projects which are available for processing by a collection of machines. Machine availability is as described for Model 1. Each bandit b evolves under processing within the finite state space E_b. The *state* of the system is a B-vector whose bth component is the state of bandit b. Write $E = \bigcup_b E_b$ for the disjoint union of the individual bandit state spaces. At each $t \in \mathbb{N}$, $|M_t|$ bandits are chosen for processing, one on each available machine. Should bandit b be chosen for processing at t when in state $i \in E_b$, then with probability P_{ij}^b it will be in state $j \in E_b$ at time $t+1$. The transition law for each bandit is Markovian and distinct bandits are assumed to evolve independently of each other. The $B - |M_t|$ bandits *not* chosen for processing at t remain stationary. We shall assume that the Markov chains determined by the B one-step transition matrices $\boldsymbol{P}^b$ (which we could observe in real time were bandit b to be processed without interruption) are irreducible and hence positive recurrent.

Admissible controls must be non-anticipative and non-idling (all machines must be fully utilised). The processing of bandit b at time t when in state $i \in E_b$ earns a positive reward $r_i \int_t^{t+1} \mathrm{e}^{-\alpha s}\,\mathrm{d}s$, where $\alpha > 0$. In the context of Model 2, we shall write $\boldsymbol{k}$ for the B-vector corresponding to the initial state. Hence as in (3) and (4) we have the total expected reward under control u given by

$$R^u(\boldsymbol{k}) = \sum_{b=1}^{B} \sum_{j \in E_b} r_j x_j^u(\boldsymbol{k}) = \sum_{j \in E} r_j x_j^u(\boldsymbol{k}), \tag{6}$$

where

$$x_j^u(\boldsymbol{k}) = \mathrm{E}_u \left\{ \int_0^\infty \mathcal{X}_j(s)\, \mathrm{e}^{-\alpha s}\,\mathrm{d}s \;\middle|\; \boldsymbol{k} \right\}, \qquad j \in E.$$

For each $t \in \mathbb{N}$, $\mathcal{X}_j(s)$, $t \leq s < t+1$, is given by

$$\mathcal{X}_j(s) = \begin{cases} 1, & \text{if at time } t \text{ bandit } b \text{ in state } j \text{ is chosen for processing, } j \in E_b,\ 1 \leq b \leq B, \\ 0, & \text{otherwise.} \end{cases}$$

The stochastic optimisation of interest is formally identical to (5).

3. Machine availability

We shall need a variety of numerical descriptors of the machine availability process $\{M_t,\ t \in \mathbb{N}\}$ and also the statement of an important technical condition for the subsequent theory. Recall that we have M machines, labelled $\{1, 2, \ldots, M\}$, and that $M_t \in 2^{\{1,2,\ldots,M\}}$ is the subset available for processing at time $t \in \mathbb{N}$. We shall suppose that the process $\{M_t,\ t \in \mathbb{N}\}$ is independent of the scheduling control and of the system state process.

We firstly introduce $2M$ derived indicator processes $\{I_{m,t},\ t \in \mathbb{N}\}$ and $\{J_{m,t},\ t \in \mathbb{N}\}$, $1 \leq m \leq M$, given by

$$I_{m,t} = \begin{cases} 1, & \text{when } m \in M_t, \\ 0, & \text{otherwise,} \end{cases}$$

$$J_{m,t} = \begin{cases} 1, & \text{when } |M_t| \geq m, \\ 0, & \text{otherwise.} \end{cases}$$

Hence $I_{m,\cdot}$ marks when machine m is available, while $J_{m,\cdot}$ indicates when at least m machines are up. It is evident that

$$J_{M,t} = \min_{1 \le m \le M} J_{m,t} = \min_{1 \le m \le M} I_{m,t}, \qquad t \in \mathbb{N}.$$

From the discrete-time processes above we derive the continuous-time extensions $\{M(s), s \in \mathbb{R}_+\}$, $\{I_m(s), s \in \mathbb{R}_+\}$, $1 \le m \le M$, and $\{J_m(s), s \in \mathbb{R}_+\}$, $1 \le m \le M$, given by

$$\begin{aligned} M(s) &= |M_t|, \qquad s \in [t, t+1),\ t \in \mathbb{N}, \\ I_m(s) &= I_{m,t}, \qquad s \in [t, t+1),\ t \in \mathbb{N}, \\ J_m(s) &= J_{m,t}, \qquad s \in [t, t+1),\ t \in \mathbb{N}. \end{aligned}$$

Plainly, from the definitions of the quantities concerned we have that

$$M(s) = \sum_{m=1}^{M} I_m(s) = \sum_{m=1}^{M} J_m(s), \qquad s \in \mathbb{R}_+,$$

and all are expressions for the number of machines available at time s. We now proceed to define the *cumulative availability processes* $\{\bar{M}(s), s \in \mathbb{R}_+\}$, $\{\bar{I}_m(s), s \in \mathbb{R}_+\}$, $1 \le m \le M$, and $\{\bar{J}_m(s), s \in \mathbb{R}_+\}$, $1 \le m \le M$, by

$$\begin{aligned} \bar{M}(s) &= \int_0^s M(u)\,\mathrm{d}u, \qquad s \in \mathbb{R}_+, \\ \bar{I}_m(s) &= \int_0^s I_m(u)\,\mathrm{d}u, \qquad s \in \mathbb{R}_+, \\ \bar{J}_m(s) &= \int_0^s J_m(u)\,\mathrm{d}u, \qquad s \in \mathbb{R}_+. \end{aligned}$$

Plainly, we have that

$$\bar{M}(s) = \sum_{m=1}^{M} \bar{I}_m(s) = \sum_{m=1}^{M} \bar{J}_m(s), \qquad s \in \mathbb{R}_+,$$

with all the above cumulative processes being non-decreasing, piecewise linear and continuous on all sample paths. The inverse process $\{\bar{M}^{-1}(s), s \in \mathbb{R}_+\}$, which is given by

$$\bar{M}^{-1}(s) = \inf_{t \in \mathbb{R}_+} \{t; \bar{M}(t) > s\}, \qquad s \in \mathbb{R}_+, \tag{7}$$

will also be non-decreasing and piecewise linear, but will have jump discontinuities corresponding to epochs at which $|M_t| = 0$. Hence $\bar{M}^{-1}(s)$ has the interpretation as the time required to achieve an accumulated machine availability equal to s. Plainly,

$$\bar{M}^{-1}(s) \ge \frac{s}{M} \to \infty, \qquad s \to \infty,$$

and we shall be concerned to ensure that the divergence of $\bar{M}^{-1}(s)$ is not worse than linear in s. The appropriate condition is given as Condition 1 below with Condition 1′ as a simpler

alternative. We define the inverse processes $\{\bar{I}_m^{-1}(s),\ s \in \mathbb{R}_+\}$, $1 \leq m \leq M$, and $\{\bar{J}_m^{-1}(s),\ s \in \mathbb{R}_+\}$, $1 \leq m \leq M$, as in (7).

To conclude this account of objects related to machine availability, we need a few further refinements of processes already described. Let μ be an integer in the range $\{0, 1, \ldots, M\}$. We develop the process $\{\bar{M}_\mu(s),\ s \in \mathbb{R}_+\}$ by

$$\bar{M}_\mu(s) = \int_0^s \{M(u) - \mu\}^+ \, \mathrm{d}u,$$

where

$$x^+ = \begin{cases} x, & \text{if } x \geq 0, \\ 0, & \text{otherwise.} \end{cases}$$

The corresponding inverse process $\{\bar{M}_\mu^{-1}(s),\ s \in \mathbb{R}_+\}$ is defined as in (7). These measures are needed in situations where μ machines are 'reserved'.

We now fix time $t \in \mathbb{R}_+$ and develop cumulative processes $\{\bar{I}_m(s;t),\ s \in \mathbb{R}_+\}$, $1 \leq m \leq M$, and $\{\bar{J}_m(s;t),\ s \in \mathbb{R}_+\}$, $1 \leq m \leq M$, starting from t. Hence, we have

$$\bar{I}_m(s;t) = \bar{I}_m(s+t) - \bar{I}_m(t), \qquad s \in \mathbb{R}_+,$$
$$\bar{J}_m(s;t) = \bar{J}_m(s+t) - \bar{J}_m(t), \qquad s \in \mathbb{R}_+.$$

For each $t \in \mathbb{R}_+$ the corresponding inverse processes $\{\bar{I}_m^{-1}(s;t),\ s \in \mathbb{R}_+\}$, $1 \leq m \leq M$, are given by

$$\bar{I}_m^{-1}(s;t) = \inf_{u \in \mathbb{R}_+} \{u;\ \bar{I}_m(u;t) > s\}, \qquad s \in \mathbb{R}_+,$$

with $\bar{I}_m^{-1}(s;t)$ having the interpretation as the time required from t until machine m's total availability exceeds s. We define $\{\bar{J}_m^{-1}(s;t),\ s \in \mathbb{R}_+\}$, $1 \leq m \leq M$, similarly.

We now state Condition 1, which is required at key points in our development. In its statement we use $\{\mathcal{F}_t,\ t \in \mathbb{N}\}$ for the filtration generated by the machine availability process $\{M_t,\ t \in \mathbb{N}\}$. Think of $\mathcal{F}_t$ as the 'up to time t' history of machine availability.

Condition 1. *There exist constants a and b such that*

$$\mathrm{E}\{\bar{J}_M^{-1}(s;t) \mid \mathcal{F}_t\} \leq a + bs, \qquad s \in \mathbb{R}_+, \text{ w.p. 1}, \tag{8}$$

for all choices of $t \in \mathbb{N}$.

If (8) is only required to hold for the case $t = 0$, we shall refer to Condition 1′.

Condition 1′. *There exist constants a and b such that*

$$\mathrm{E}\{\bar{J}_M^{-1}(s) \mid \mathcal{F}_0\} \leq a + bs, \qquad s \in \mathbb{R}_+, \text{ w.p. 1}. \tag{9}$$

Note that, from the definitions of the quantities concerned, it is always true that $M(s) \geq MJ_M(s)$, $s \in \mathbb{R}_+$. Inequality (9) is then seen to imply that

$$\mathrm{E}\{\bar{M}^{-1}(s) \mid \mathcal{F}_0\} \leq a + \left(\frac{b}{M}\right) s, \qquad s \in \mathbb{R}_+, \text{ w.p. 1}.$$

We now give examples of machine availability processes for which Condition 1 or 1′, as appropriate, hold.

Example 1. Suppose that the process of the number of machines available $\{|M_t|,\ t \in \mathbb{N}\}$ is an irreducible (and hence positive recurrent) Markov chain. To see that Condition 1 is then satisfied, observe that when $s \in \mathbb{N}$

$$\mathrm{E}\{\bar{J}_M^{-1}(s;t) \,|\, |M_t| = m\} = \mathrm{E}\left(T_{mM} + \sum_{\nu=1}^{s} T_{MM}(\nu)\right) \tag{10}$$

$$= \mu_{mM} + s\mu_{MM}$$

$$\leq \left(\max_{1\leq m\leq M} \mu_{mM}\right) + s\mu_{MM}. \tag{11}$$

In (10), T_{mM} is the time from t until the first occasion upon which all M machines are available, while the $T_{MM}(\nu)$, $1 \leq \nu \leq s$, are successive first return times to this state. The parameters μ_{mM} and μ_{MM} are the corresponding (finite) expectations of these random times. From (11) and the Markovian structure, it is clear that Condition 1 is satisfied.

Example 2. Suppose that machine m has breakdown and repair rates equal to β_m and ρ_m respectively in the sense that

$$p(I_{m,t+1} = 0 \,|\, I_{m,t} = 1) = \beta_m, \qquad t \in \mathbb{N},$$
$$p(I_{m,t+1} = 1 \,|\, I_{m,t} = 0) = \rho_m, \qquad t \in \mathbb{N},$$

for $1 \leq m \leq M$, and that the breakdown/repair processes of different machines are independent. If $0 < \beta_m, \rho_m < 1$, $1 \leq m \leq M$, then $\{M_t,\ t \in \mathbb{N}\}$ is an irreducible Markov chain. An analysis similar to that applied in Example 1 to (11) yields the conclusion that Condition 1 is satisfied.

Example 3. We have an alternating renewal process $\{A_1, B_1, A_2, B_2, \ldots\}$ with inter-arrival times having finite means and taking values in the positive integers. During the A-periods $([0, A_1) \cup [A_1 + B_1, A_1 + B_1 + A_2) \cup \cdots)$ all the machines are switched off, while during the B-periods $([A_1, A_1 + B_1) \cup [A_1 + B_1 + A_2, A_1 + B_1 + A_2 + B_2) \cup \cdots)$ machine availability evolves according to one of the models in Example 1 or 2. It is straightforward to demonstrate that Condition 1 is satisfied.

Example 4. Consider any process $\{|M_t|,\ t \in \mathbb{N}\}$ for which $|M_t|$ is non-decreasing almost surely. Plainly, in this case,

$$\bar{J}_M^{-1}(s) = s + X_M, \quad \text{where } X_M = \inf_{t\in\mathbb{R}_+} \{t; |M_t| = M\}$$

is the first occasion at which all M machines are available for processing. Condition $1'$ will be satisfied if $\mathrm{E}(X_M \,|\, \mathcal{F}_0)$ is uniformly bounded. Please note that, henceforth, we shall *not* make conditioning upon $\mathcal{F}_0$ explicit in the notation.

4. Tools of analysis for the scheduling models

In both Models 1 and 2, E represents the universal set of objects to be scheduled on the available machines. In both cases we shall define a vector of coefficients $\boldsymbol{A}^S = \{A_i^S\}_{i\in E}$ for each subset $S \subseteq E$. The details of these vectors in the two cases are given below. For now, we simply note that $A_i^S > 0$, $i \in S$ and $A_i^S = 0$, $i \notin S$. A key idea in our analyses is that, in both models, the expected rewards under control u,

$$R^u(\boldsymbol{k}) = \sum_{j\in E} r_j x_j^u(\boldsymbol{k}),$$

may be expressed in terms of the quantities

$$A^u(S, \boldsymbol{k}) = \sum_{j \in S} A_j^S x_j^u(\boldsymbol{k}), \qquad S \subseteq E.$$

In outline, this works as follows: a so-called *adaptive greedy algorithm* $\mathrm{AG}(\boldsymbol{A}, \boldsymbol{r})$ has inputs given by the collection $\boldsymbol{A} = \{\boldsymbol{A}^S, S \subseteq E\}$ together with the reward vector $\boldsymbol{r} = \{r_j, j \in E\}$. The outputs from the algorithm are a set of non-negative real numbers G_j, $j \in E$, called *Gittins indices*. See Katehakis and Veinott (1987) for an alternative approach to the computation of these indices. We shall always assume that members of E are labelled $\{1, 2, \ldots, |E|\}$ such that

$$G_{|E|} \geq G_{|E|-1} \geq \cdots \geq G_2 \geq G_1.$$

In none of the results in Sections 4–6 does it matter how ties are broken between options whose index values are equal. Note that a *Gittins index scheduling policy* is one which always chooses from among the jobs or bandits available for processing those with highest index values. Such policies are optimal in the special single machine cases of Models 1 and 2 with $|M_t| = 1, t \in \mathbb{N}$. Write $S_j = \{j, j-1, \ldots, 1\}$ for the subset of E of cardinality j with lowest Gittins indices. We shall not require further details of the algorithm $\mathrm{AG}(\boldsymbol{A}, \boldsymbol{r})$ and the associated indices, beyond the key fact that the structure of $\mathrm{AG}(\boldsymbol{A}, \boldsymbol{r})$ is such that the rewards $\{r_i, i \in E\}$ may be expressed as

$$r_i = G_{|E|} A_i^E - \sum_{j=i}^{|E|-1} (G_{j+1} - G_j) A_i^{S_j}, \tag{12}$$

where it will emerge that $A_i^E = 1$, $i \in E$. For more details regarding Gittins indices and the adaptive greedy algorithm, the reader is referred to Bertsimas and Niño-Mora (1996).

Lemma 1 is a straightforward consequence of (3), (6) and (12).

Lemma 1. *For both Models 1 and 2 and all initial states* $\boldsymbol{k}$,

$$R^u(\boldsymbol{k}) = G_{|E|} \mathrm{E}\left\{ \int_0^\infty M(s)\, \mathrm{e}^{-\alpha s}\, \mathrm{d}s \right\} - \sum_{j=1}^{|E|-1} (G_{j+1} - G_j) A^u(S_j, \boldsymbol{k}).$$

Lemma 1 leads on immediately to Corollary 1, which is the key tool for the evaluation of proposed controls for our scheduling models. We firstly introduce

$$A(S, \boldsymbol{k}) = \inf_{u \in \mathcal{U}} A^u(S, \boldsymbol{k}), \qquad S \subseteq E.$$

Corollary 1. (Evaluating admissible controls.) (i) *For both Models 1 and 2 and all initial states* $\boldsymbol{k}$,

$$R^{\mathrm{opt}}(\boldsymbol{k}) \leq G_{|E|} \mathrm{E}\left\{ \int_0^\infty M(s)\, \mathrm{e}^{-\alpha s}\, \mathrm{d}s \right\} - \sum_{j=1}^{|E|-1} (G_{j+1} - G_j) A(S_j, \boldsymbol{k});$$

(ii) *For both Models 1 and 2, all admissible controls* u *and all initial states* $\boldsymbol{k}$

$$R^{\mathrm{opt}}(\boldsymbol{k}) - R^u(\boldsymbol{k}) \leq \sum_{j=1}^{|E|-1} (G_{j+1} - G_j)\{A^u(S_j, \boldsymbol{k}) - A(S_j, \boldsymbol{k})\}.$$

The practical challenge from Corollary 1(ii) in obtaining bounds on the degree of reward suboptimality of some given heuristic control u is in deriving tight upper bounds for the associated quantities $A^u(S_j, \boldsymbol{k})$ and lower bounds for the $A(S_j, \boldsymbol{k})$. We shall now proceed to describe in greater detail how the above ideas apply to our two scheduling models. In particular, by the conclusion of this section we will have obtained suitable lower bounds for the quantities $A(S_j, \boldsymbol{k})$ for both Models 1 and 2.

4.1. Model 1: discounted branching bandit

Before proceeding to develop effective lower bounds for $A(S_j, \boldsymbol{k})$ for Model 1, we must first describe the vectors $\boldsymbol{A}^S$ required for the analysis. Fix job class $i \in E$ and subset $S \subseteq E$ with $i \in S$. Consider the discounted branching bandit model above, but in a set-up in which only a single job of class i is present at time 0, that processing is provided by a single machine which is always available ($|M_t| = 1,\ t \in \mathbb{N}$) and that a control operates which gives jobs from classes in S^c priority over those in S. We write $T_i(S^c)$ for the first time at or after time 1 at which no S^c-jobs are present in the system. Note that it is clear that the distribution of the random variable $T_i(S^c)$ is independent of which control from the specified class ($S^c \to S$) is utilised. Also note that under the stability condition given for Model 1 in Section 2, we may assume that whenever $0 \in S$ all positive moments of $T_i(S^c)$ are finite.

With the above in place, we follow Bertsimas and Niño-Mora (1996) in their analysis of the single machine case ($|M_t| = 1,\ t \in \mathbb{N}$) in defining the vector $\boldsymbol{A}^S$ for subsets of E containing 0 as follows:

$$A_i^S = \begin{cases} \dfrac{1 - \mathrm{E}(\mathrm{e}^{-\alpha T_i(S^c)})}{1 - \mathrm{e}^{-\alpha}}, & i \in S, \\ 0, & i \notin S. \end{cases}$$

We shall now prove the following intuitive result in relation to Model 1.

Lemma 2. *The idling class* 0 *has the smallest Gittins index.*

Proof. We firstly observe that in the single machine case ($|M_t| = 1,\ t \in \mathbb{N}$) of Model 1 a Gittins index policy is optimal in the class of all non-anticipative controls. See Bertsimas and Niño-Mora (1996). We shall now suppose that the index for job 0 is strictly greater than those for classes in some non-empty subset $I \subseteq E \setminus \{0\}$, but less than or equal to those for classes in $E \setminus [I \cup \{0\}]$ and obtain a contradiction.

Consider a single machine problem in which at time $t = 0$, job 0 is present along with a single job i^* from a class in I. Under our hypothesis the optimal control will be a Gittins index policy in which 0 will be scheduled at time 0. Thereafter jobs from $E \setminus I$ will be processed according to the values of their indices. In particular, since job 0 is always present, the job i^* will never be scheduled for processing. Since no reward is earned from the processing at time 0, we write the expected reward from this control as $\mathrm{e}^{-\alpha} R$.

Now consider the control which schedules job i^* for processing at time 0 and which from time $t = 1$ onwards schedules the external arrivals during $[0, 1)$ together with job 0 and their descendants according to the values of their Gittins indices. In particular, the internal descendants of job i^* at time $t = 1$ are laid aside and never scheduled. Since from time $t = 1$, this control yields outcomes which are stochastically identical to those from the Gittins index policy of the previous paragraph, it is clear that the expected reward from this control is $r_{i^*} + \mathrm{e}^{-\alpha} R$. This contradicts the optimality of the Gittins index policy. The result follows.

It is an immediate consequence of Lemma 2 that we may assume without loss of generality that idling class 0 is a member of S_j for all j, $1 \leq j \leq |E|$. We can then conclude that for $i \in S_j$ all positive moments of $T_i(S_j^c)$ are finite.

Before proceeding to the development of lower bounds for $A(S_j, \boldsymbol{k})$ in Theorems 1 and 2, we need one further piece of notation. For any fixed subset $S \subseteq E$, we consider a discounted branching bandit in some initial state $\boldsymbol{k}$ under a control $S^c \to S$ which gives priority to classes in S^c over those in S on a single machine ($|M_t| = 1$, $t \in \mathbb{N}$). We write $T_{\boldsymbol{k}}(S^c)$ for the first time at which no S^c-jobs are present. If no S^c-jobs are present in $\boldsymbol{k}$ then $T_{\boldsymbol{k}}(S^c) = 0$. It is clear that the distribution of $T_{\boldsymbol{k}}(S^c)$ is independent of which control from the specified class ($S^c \to S$) is utilised.

Lemma 3. *When $\mathcal{U}$ is the class of all non-anticipative controls for Model 1,*

$$A(S_j, \boldsymbol{k}) \geq \mathrm{E}\left\{\int_{M^{-1}(T_{\boldsymbol{k}}(S_j^c))}^{\infty} M(s)\, \mathrm{e}^{-\alpha s}\, \mathrm{d}s\right\}\left(1 - \frac{\alpha\, \mathrm{e}^{2\alpha}}{2}\left[\max_{i \in S_j} \mathrm{E}\{(T_i(S_j^c))^2\}\right]\right), \quad 1 \leq j \leq |E|, \tag{13}$$

for all initial states $\boldsymbol{k}$.

Proof. It follows from the definitions of the quantities involved that

$$\begin{aligned} A(S_j, \boldsymbol{k}) &= \inf_{u \in \mathcal{U}} \sum_{i \in S_j} A_i^{S_j} x_i^u(\boldsymbol{k}) \\ &= \inf_{u \in \mathcal{U}} \sum_{i \in S_j} A_i^{S_j}\, \mathrm{E}_u\left(\sum_{l=1}^{\infty} \int_{\nu_{i,l}}^{\nu_{i,l}+1} \mathrm{e}^{-\alpha s}\, \mathrm{d}s \,\middle|\, \boldsymbol{k}\right) \\ &= \inf_{u \in \mathcal{U}} \mathrm{E}_u\left(\sum_{i \in S_j} \sum_{l=1}^{\infty} \mathrm{e}^{-\alpha \nu_{i,l}} \int_0^{T_{i,l}(S_j^c)} \mathrm{e}^{-\alpha s}\, \mathrm{d}s \,\middle|\, \boldsymbol{k}\right). \end{aligned} \tag{14}$$

In the right-hand expression in (14), $\nu_{i,l}$ denotes the time of the lth occasion (for any suitable numbering of such occasions) upon which the control u processes a class i job. The collection $\{T_{i,l}(S_j^c), l \in \mathbb{Z}_+\}$ is of i.i.d. random variables, all of which share the distribution of $T_i(S_j^c)$ above. We shall obtain the lower bound in (13) by relaxing the minimisation in (14).

The relaxation will take the form of a single machine scheduling problem as follows: consider the system with initial state $\boldsymbol{k}$ and processed by a single machine which is always available ($|M_t| = 1, t \in \mathbb{N}$). Denote by $\tilde{\mathcal{U}}$ the set of non-anticipative controls and consider the stochastic optimisation problem given by

$$\tilde{A}(S_j, \boldsymbol{k}) = \inf_{u \in \tilde{\mathcal{U}}} \mathrm{E}_u\left[\sum_{i \in S_j} \sum_{l=1}^{\infty} \exp\{-\alpha \lceil \bar{M}^{-1}(\tilde{\nu}_{i,l}) \rceil\} \int_0^{T_{i,l}(S_j^c)} \mathrm{e}^{-\alpha s}\, \mathrm{d}s \,\middle|\, \boldsymbol{k}\right]. \tag{15}$$

In (15), $\tilde{\nu}_{i,l}$ denotes the time of the lth occasion upon which the single machine controller processes a class i job, the $\{T_{i,l}(S_j^c),\ l \in \mathbb{Z}_+\}$ are as in the previous paragraph and $\lceil x \rceil$ is the integer part of x. To see that (15) is a relaxation of (14), note that in the objective in (15) rewards accruing from machine allocations at times $t = \bar{M}(s), \bar{M}(s)+1, \ldots, \bar{M}(s+1)-1$ attract discounting $\mathrm{e}^{-\alpha s}$ for any $s \in \mathbb{N}$ and hence can be thought of as accruing at time s in problem (14). Further, all controls in $\mathcal{U}$ for (14) may be mapped in an obvious way to controls in $\tilde{\mathcal{U}}$ for (15). It follows that for all initial states $\boldsymbol{k}$ and all j, $1 \leq j \leq |E|$,

$$A(S_j, \boldsymbol{k}) \geq \tilde{A}(S_j, \boldsymbol{k}). \tag{16}$$

The single machine problem (15) is easily solved. It follows from a simple pairwise interchange argument which utilises the fact that $\{\bar{M}^{-1}(s),\ s \in \mathbb{R}_+\}$ is non-decreasing on all sample paths, that the infimum in (15) will be achieved by a control u_j, say, which gives priority to S_j^c over S_j. Under u_j, the first epoch at which an S_j-job is chosen in the single machine problem is $T_{\boldsymbol{k}}(S_j^c)$. Suppose that the S_j-job chosen for processing by u_j at time $T_{\boldsymbol{k}}(S_j^c)$ is in class $i \in S_j$. Under u_j, the processing of i at $T_{\boldsymbol{k}}(S_j^c)$ will be followed by an S_j^c-excursion (i.e. a period during which only S_j^c-jobs are processed) of length $T_{i,1}(S_j^c)$. Under the objective in (15) the contribution to $\tilde{A}(S_j, \boldsymbol{k})$ from the processing of $i \in S_j$ at $T_{\boldsymbol{k}}(S_j^c)$ is given by

$$\begin{aligned}
&\mathrm{E}\Bigg[\exp\{-\alpha\lceil\bar{M}^{-1}(T_{\boldsymbol{k}}(S_j^c))\rceil\}\int_0^{T_{i,1}(S_j^c)} \mathrm{e}^{-\alpha s}\,\mathrm{d}s\Bigg] \\
&\quad\geq \mathrm{E}\left[\exp\{-\alpha\lceil\bar{M}^{-1}(T_{\boldsymbol{k}}(S_j^c))\rceil\}\left\{T_{i,1}(S_j^c) - \frac{\alpha}{2}(T_{i,1}(S_j^c))^2\right\}\right] \\
&\quad\geq \mathrm{E}\Bigg(\sum_{t=T_{\boldsymbol{k}}(S_j^c)}^{T_{\boldsymbol{k}}(S_j^c)+T_{i,1}(S_j^c)-1} \int_{\lceil\bar{M}^{-1}(t)\rceil}^{\lceil\bar{M}^{-1}(t)\rceil+1} \mathrm{e}^{-\alpha s}\,\mathrm{d}s\Bigg) \\
&\qquad - \frac{\alpha}{2}\,\mathrm{E}[\exp\{-\alpha\lceil\bar{M}^{-1}(T_{\boldsymbol{k}}(S_j))\rceil\}]\,\mathrm{E}\{(T_{i,1}(S_j^c))^2\} \qquad (17)
\end{aligned}$$

and the next decision epoch at which an S_j-job is chosen for processing is $T_{\boldsymbol{k}}(S_j^c) + T_{i,1}(S_j^c)$. Repeating the calculation to (17) for successive decision epochs and aggregating, we obtain that

$$\begin{aligned}
\tilde{A}(S_j, \boldsymbol{k}) &\geq \mathrm{E}\Bigg(\sum_{t=T_{\boldsymbol{k}}(S_j^c)}^{\infty} \int_{\lceil\bar{M}^{-1}(t)\rceil}^{\lceil\bar{M}^{-1}(t)\rceil+1} \mathrm{e}^{-\alpha s}\,\mathrm{d}s\Bigg) \\
&\quad - \frac{\alpha}{2}\,\mathrm{E}_{u_j}\left[\sum_{i\in S_j}\sum_{l=1}^{\infty}\exp\{-\alpha\lceil\bar{M}^{-1}(\tilde{\nu}_{i,l})\rceil\}\;\middle|\;\boldsymbol{k}\right]\left[\max_{i\in S_j}\mathrm{E}\{(T_i(S_j^c))^2\}\right] \qquad (18) \\
&\geq \mathrm{E}\Bigg(\sum_{t=T_{\boldsymbol{k}}(S_j^c)}^{\infty} \int_{\lceil\bar{M}^{-1}(t)\rceil}^{\lceil\bar{M}^{-1}(t)\rceil+1} \mathrm{e}^{-\alpha s}\,\mathrm{d}s\Bigg)\left(1 - \frac{\alpha\,\mathrm{e}^{\alpha}}{2}\left[\max_{i\in S_j}\mathrm{E}\{(T_i(S_j^c))^2\}\right]\right) \qquad (19) \\
&\geq \mathrm{E}\left\{\int_{\bar{M}^{-1}(T_{\boldsymbol{k}}(S_j^c))}^{\infty} M(s)\,\mathrm{e}^{-\alpha s}\,\mathrm{d}s\right\}\left(1 - \frac{\alpha\,\mathrm{e}^{2\alpha}}{2}\left[\max_{i\in S_j}\mathrm{E}\{(T_i(S_j^c))^2\}\right]\right). \qquad (20)
\end{aligned}$$

Please note that (19) follows from (18) by bounding the second term in the latter by using the inclusion

$$\{\tilde{\nu}_{i,l};\ i \in S_j,\ l \in \mathbb{Z}^+\} \subseteq \{t; t \in \mathbb{Z}^+ \text{and } t \geq T_{\boldsymbol{k}}(S_j^c)\}.$$

Inequality (20) is deduced from (19) by straightforward analysis and by the definitions of the quantities involved. The result is now an immediate consequence of (16) and (20).

For Model 1 we now consider a more restrictive class of admissible controls and, consequentially, a higher lower bound for $A(S_j, \boldsymbol{k})$. This is the class of *local* controls which have the property that, if a class i job is assigned to some machine m, say, at some time t, then the $\boldsymbol{n}_i^{tm} = \{n_{i1}^{tm}, n_{i2}^{tm}, \ldots, n_{iC}^{tm}\}$ jobs of classes $1, 2, \ldots, C$ which replace it at time $t+1$ must also be processed on machine m. Hence jobs are processed on the same machine as all of their

descendants. Such classes of controls for Model 1 are relevant in the context of load balancing for distributed multi-class service systems.

Lemma 4. *When $\mathcal{U}$ is the class of all non-anticipative local controls for Model 1,*

$$A(S_j, \boldsymbol{k}) \geq \mathrm{E}\left\{\int_{M^{-1}(T_{\boldsymbol{k}}(S_j))}^{\infty} M(s)\,\mathrm{e}^{-\alpha s}\,\mathrm{d}s\right\}, \qquad 1 \leq j \leq |E|,$$

for all initial states $\boldsymbol{k}$.

Proof. Developing (14) above, we have that

$$A(S_j, \boldsymbol{k}) = \inf_{u\in\mathcal{U}} \mathrm{E}_u\left(\sum_{m=1}^{M}\sum_{i\in S_j}\sum_{l=1}^{\infty} \mathrm{e}^{-\alpha v_{i,m,l}} \int_0^{T_{i,m,l}(S_j^{\mathrm{c}})} \mathrm{e}^{-\alpha s}\,\mathrm{d}s \;\middle|\; \boldsymbol{k}\right) \tag{21}$$

$$\geq \inf_{u\in\mathcal{U}} \mathrm{E}_u\left(\sum_{m=1}^{M}\int_0^{\infty} \xi_m^j(s)\,\mathrm{e}^{-\alpha s}\,\mathrm{d}s \;\middle|\; \boldsymbol{k}\right). \tag{22}$$

In (21), $v_{i,m,l}$ denotes the time of the lth occasion upon which the control u processes a job from class $i \in S_j$ on machine m. With each $v_{i,m,l}$ is associated a random variable $T_{i,m,l}(S_j^{\mathrm{c}})$ to be thought of as the number of S_j^{c}-descendants generated by the allocation at time $v_{i,m,l}$. In (22) we define for each s, $t \leq s < t+1$, with $t \in \mathbb{N}$,

$$\xi_m^j(s) = \begin{cases} 1, & \text{if machine } m \text{ processes an } S_j\text{-job or an } S_j^{\mathrm{c}}\text{-descendant} \\ & \text{of a previously scheduled } S_j\text{-job at time } t; \\ 0, & \text{otherwise.} \end{cases}$$

Inequality (22) follows from (21) and the observation that the S_j^{c}-processing generated by the allocation of $i \in S_j$ at time $v_{i,m,l}$ on machine m will be scheduled under local control u during $[v_{i,m,l},\, v_{i,m,l} + T_{i,m,l}(S_j^{\mathrm{c}}))$ at the earliest. As in the proof of Lemma 3, we consider a single machine relaxation of (22) in which a reward accruing in the single machine problem at time t attracts discounting $\exp\{-\alpha\lceil M^{-1}(t)\rceil\}$, $t \in \mathbb{N}$. Hence we have

$$A(S_j, \boldsymbol{k}) \geq \inf_{u\in\tilde{\mathcal{U}}} \mathrm{E}_u\left[\sum_{t=0}^{\infty} \xi^j(t) \int_{\lceil \tilde{M}^{-1}(t)\rceil}^{\lceil \tilde{M}^{-1}(t)\rceil+1} \mathrm{e}^{-\alpha s}\,\mathrm{d}s \;\middle|\; \boldsymbol{k}\right], \tag{23}$$

where $\tilde{\mathcal{U}}$ is the set of non-anticipative controls for the single machine problem and for each $t \in \mathbb{N}$,

$$\xi^j(t) = \begin{cases} 1, & \text{if an } S_j\text{-job or an } S_j^{\mathrm{c}}\text{-descendant of a previously scheduled } S_j\text{-job} \\ & \text{is processed at time } t; \\ 0, & \text{otherwise.} \end{cases}$$

The single machine problem in (23) is solved by a control u_j, say, which gives priority to S_j^{c} over S_j. For such a control,

$$\xi^j(t) = 1 \quad \Longleftrightarrow \quad t \geq T_{\boldsymbol{k}}(S_j^{\mathrm{c}}),$$

and we conclude that

$$A(S_j, \boldsymbol{k}) \geq \mathrm{E}\Bigg(\sum_{t=T_{\boldsymbol{k}}(S_j^{\mathrm{c}})}^{\infty} \int_{\lceil \bar{M}^{-1}(t) \rceil}^{\lceil \bar{M}^{-1}(t) \rceil + 1} \mathrm{e}^{-\alpha s} \, \mathrm{d}s \Bigg) \geq \mathrm{E}\Bigg\{ \int_{\bar{M}^{-1}(T_{\boldsymbol{k}}(S_j^{\mathrm{c}}))}^{\infty} M(s) \, \mathrm{e}^{-\alpha s} \, \mathrm{d}s \Bigg\},$$

as required.

4.2. Model 2: discounted multi-armed bandit

As with Model 1, we begin our discussion of lower bounds for $A(S_j, \boldsymbol{k})$ in the context of Model 2 by developing the appropriate choice of vector $\boldsymbol{A}^S$. Fix $i \in E_b \subseteq E$ and subset $S \subseteq E$ with $i \in S$. Consider a set-up in which only bandit b in state i is present at time 0. Under processing by a single machine ($|M_t| = 1$, $t \in \mathbb{N}$), bandit b evolves as a Markov chain with one-step transition matrix $\boldsymbol{P}^b$. We write $T_i(S^{\mathrm{c}})$ for the first time at or after time 1 at which bandit b enters S. It follows from the positive recurrence of the Markov chain that all positive moments of $T_i(S^{\mathrm{c}})$ are finite. We now follow Bertsimas and Niño-Mora (1996) in their analysis of the single machine case ($|M_t| = 1, t \in \mathbb{N}$) in defining the matrix $\boldsymbol{A}^S$ as follows:

$$A_i^S = \begin{cases} \dfrac{1 - \mathrm{E}(\mathrm{e}^{-\alpha T_i(S^{\mathrm{c}})})}{1 - \mathrm{e}^{-\alpha}}, & i \in S, \\ 0, & i \notin S. \end{cases}$$

Recall that members of E are numbered such that

$$G_{|E|} \geq G_{|E|-1} \geq \cdots \geq G_2 \geq G_1.$$

We write

$$B_j = \{b \in B;\ E_b \cap S_j = \varnothing\}, \qquad 1 \leq j \leq |E|,$$

for the collection of bandits whose state spaces have no intersection with $S_j = \{j, j-1, \ldots, 2, 1\}$, the set of states with the j lowest indices. We also write

$$\mu_j = |B_j|, \qquad 1 \leq j \leq |E|.$$

It is evident that if $\mu_j \geq M$, then there exist admissible controls which *never* schedule members of S_j. Hence it must be true for all initial states $\boldsymbol{k}$ that

$$\mu_j \geq M \quad \Longrightarrow \quad A(S_j, \boldsymbol{k}) = 0. \tag{24}$$

Now fix j, $1 \leq j \leq |E|$. Consider the operation of a control which processes those bandits *not* in B_j only and which gives S_j^{c} priority over S_j. Let the initial state be $\boldsymbol{k}$. We write $T_{\boldsymbol{k}}(S_j^{\mathrm{c}})$ for the first epoch at or after time zero at which all the bandits being processed (i.e. those not in B_j) have states in S_j. We can now state Lemma 5 which extends (24) to give effective lower bounds on $A(S_j, \boldsymbol{k})$ for all choices of j.

Lemma 5. *When $\mathcal{U}$ is the class of non-anticipative, non-idling controls for Model 2,*

$$A(S_j, \boldsymbol{k}) \geq \mathrm{E}\Bigg[\int_{\bar{M}_{\mu_j}^{-1}(T_{\boldsymbol{k}}(S_j^{\mathrm{c}}))}^{\infty} \{M(s) - \mu_j\}^+ \, \mathrm{e}^{-\alpha s} \, \mathrm{d}s \Bigg], \qquad 1 \leq j \leq |E|,$$

for all initial states $\boldsymbol{k}$.

Proof. By reasoning similar to that which yields (22) in the proof of Lemma 4, we have that

$$A(S_j, \boldsymbol{k}) \geq \inf_{u \in \mathcal{U}} \mathrm{E}_u \Bigg\{ \sum_{m=1}^{M} \sum_{b=1}^{B} \int_0^{\infty} \xi_{mb}^j(s) \, \mathrm{e}^{-\alpha s} \, \mathrm{d}s \;\Bigg|\; \boldsymbol{k} \Bigg\}, \tag{25}$$

where for each s, $t \leq s < t+1$, with $t \in \mathbb{N}$,

$$\xi^j_{mb}(s) = \begin{cases} 1, & \text{if machine } m \text{ processes bandit } b \text{ at time } t, \text{ where } b \\ & \text{has paid its first visit to } S_j \text{ at some time at or before } t; \\ 0, & \text{otherwise.} \end{cases}$$

As in the proofs of Lemmas 3 and 4, we bound $A(S_j, \boldsymbol{k})$ below by means of a single machine relaxation of the stochastic optimisation problem in (25). Again, rewards accruing in the single machine problem at time t attract discounting $\exp\{-\alpha\lceil \tilde{M}^{-1}(t)\rceil\}$, $t \in \mathbb{N}$. For suitable choice of admissible controls $\tilde{\mathcal{U}}$ for the single machine problem (specified below) we have

$$A(S_j, \boldsymbol{k}) \geq \inf_{u \in \tilde{\mathcal{U}}} \mathrm{E}_u \left[\sum_{t=0}^{\infty} \xi^j(t) \int_{\lceil \tilde{M}^{-1}(t)\rceil}^{\lceil \tilde{M}^{-1}(t)\rceil + 1} \mathrm{e}^{-\alpha s} \, \mathrm{d}s \;\middle|\; \boldsymbol{k} \right] \tag{26}$$

with for each $t \in \mathbb{N}$,

$$\xi^j(t) = \begin{cases} 1, & \text{if a bandit is processed at time } t, \text{ which has paid} \\ & \text{its first visit to } S_j \text{ at some time at or before } t; \\ 0, & \text{otherwise.} \end{cases}$$

The admissible controls $\tilde{\mathcal{U}}$ are non-anticipative and non-idling with the additional feature that the μ_j bandits whose state spaces have no intersection with S_j may only be scheduled at certain decision epochs. The idea here is that these bandits can never contribute to the expression on the right-hand side of (25) and hence should be scheduled whenever possible. Number these bandits $1, 2, \ldots, \mu_j$. In the single machine relaxation, we restrict to controls for which bandit $b \in \{1, 2, \ldots, \mu_j\}$ may only be processed at times $\tilde{M}(t) + \{b-1\}^+$ for those $t \in \mathbb{N}$ for which $|M_t| \geq b$. At all other epochs the single machine relaxation makes a free choice from bandits $\{\mu_j + 1, \mu_j + 2, \ldots, B\}$. With this choice of $\tilde{\mathcal{U}}$, it is straightforward to establish that the infimum in (26) is achieved by a control which

(i) schedules $b \in \{1, 2, \ldots, \mu_j\}$ at all qualifying epochs; and

(ii) makes all free choices by giving S_j^{c} priority over S_j.

Evaluating the right-hand side of (26) under such a control u_j, say, we have

$$\begin{aligned} A(S_j, \boldsymbol{k}) &\geq \mathrm{E}\left[\sum_{t;\,|M_t| \geq \mu_j + 1} \sum_{s=\tilde{M}(t)+\mu_j}^{\tilde{M}(t+1)-1} \int_{\lceil \tilde{M}^{-1}(s)\rceil}^{\lceil \tilde{M}^{-1}(s)\rceil + 1} \mathrm{e}^{-\alpha v} \, \mathrm{d}v \;\middle|\; \boldsymbol{k} \right] \\ &\geq \mathrm{E}\left[\int_{\tilde{M}_{\mu_j}^{-1}(T_{\boldsymbol{k}}(S_j^{\mathrm{c}}))}^{\infty} \{M(s) - \mu_j\}^+ \, \mathrm{e}^{-\alpha s} \, \mathrm{d}s \right] \end{aligned}$$

as required.

It may assist the reader at this stage to review Corollary 1 and the text following. For the goal of assessing heuristic policies for our scheduling models we have progressed via Lemmas 3–5 to the point of having suitable lower bounds for the quantities $A(S_j, \boldsymbol{k})$. We take the analysis further in the next two sections by proposing index-based controls u and deriving upper bounds on the resulting $A^u(S_j, \boldsymbol{k})$. We will then be in a position to utilise Corollary 1 to obtain bounds on the degree of reward suboptimality of the controls of interest.

5. On the evaluation of a class of index-based controls for discounted branching bandits

In Model 1, the initial state $\boldsymbol{k}$ describes the collection of jobs which (along with the M class 0 jobs) are present in the system at $t = 0$. We shall abuse notation by identifying $\boldsymbol{k}$ with this collection. Now, consider a partition of collection $\boldsymbol{k}$ given by

$$\boldsymbol{k} = \bigcup_{m=1}^{M} \hat{\boldsymbol{k}}(m)$$

where the notation indicates that the jobs in collection $\hat{\boldsymbol{k}}(m)$ (and their descendants) will be associated with machine m, $1 \leq m \leq M$. This partition will be denoted $\hat{\boldsymbol{k}}$ in what follows. Recall that a single class 0 job is allocated to each machine. Use 0_m for the one allocated to machine m, $1 \leq m \leq M$. Corresponding to partition $\hat{\boldsymbol{k}}$, we develop an index-based control $u_G(\hat{\boldsymbol{k}})$ which is structured as follows:

- at each time $t \in \mathbb{N}$ for which $I_m(t) = 1$, machine m processes a job chosen from $\{0_m\}$ and the descendants of the jobs in collection $\hat{\boldsymbol{k}}(m)$ which is
  - (a) present in the system, and
  - (b) has maximal Gittins index.

Note that this control is both non-anticipative and local. It simply divides up the jobs between the machines at time 0, after which each machine processes the resulting descendants according to an index policy.

Consider now the collection $\hat{\boldsymbol{k}}(m) \cup \{0_m\}$ evolving from time $t = 0$ under a control which gives S^c priority over S with processing provided by a single permanently available machine ($|M_t| = 1, t \in \mathbb{N}$). In such a set-up, we shall use $T_{\hat{\boldsymbol{k}}(m)}(S^c)$ for the first time at which no S^c-jobs are present. Further, in the statement of Lemma 6 we shall require the constants

$$K_1(S_j) = \alpha\, \mathrm{e}^{\alpha} \max_{i \in S_j} [a\, \mathrm{E}(T_i(S_j^c)) + (b-1)\, \mathrm{E}\{(T_i(S_j^c))^2\}], \qquad 1 \leq j \leq |E|, \tag{27}$$

with a and b as in (8).

Lemma 6. *For Model 1, with the machine availability process satisfying Condition 1,*

$$A^{u_G(\hat{\boldsymbol{k}})}(S_j, \boldsymbol{k}) \leq \{1 + K_1(S_j)\}\, \mathrm{E}\left(\sum_{m=1}^{M} \left[\int_{\tilde{I}_m^{-1}\{T_{\hat{\boldsymbol{k}}(m)}(S_j^c)\}}^{\infty} I_m(s)\, \mathrm{e}^{-\alpha s}\, \mathrm{d}s \right] \right), \qquad 1 \leq j \leq |E|,$$

for all initial states $\boldsymbol{k}$ and associated partitions $\hat{\boldsymbol{k}}$.

Proof. We use $A_m^{u_G(\hat{\boldsymbol{k}})}(S_j, \boldsymbol{k})$ to denote the contribution to $A^{u_G(\hat{\boldsymbol{k}})}(S_j, \boldsymbol{k})$ from the processing on machine m under control $u_G(\hat{\boldsymbol{k}})$. Utilising the notation in the proof of Lemma 4, we have that

$$A_m^{u_G(\hat{\boldsymbol{k}})}(S_j, \boldsymbol{k}) = \mathrm{E}_{u_G(\hat{\boldsymbol{k}})} \left\{ \sum_{i \in S_j} \sum_{l=1}^{\infty} \mathrm{e}^{-\alpha v_{i,m,l}} \int_0^{T_{i,m,l}(S_j^c)} \mathrm{e}^{-\alpha s}\, \mathrm{d}s \;\middle|\; \hat{\boldsymbol{k}}(m) \right\}, \qquad 1 \leq m \leq M. \tag{28}$$

We now rewrite (28) and utilise the facts that $e^{\alpha t} \leq 1+\alpha t\, e^{\alpha t}, \alpha \geq 0, t \geq 0$ and that $\bar{I}_m^{-1}(s;t)-s$ is non-negative and non-decreasing in s for all $t \geq 0$ to obtain that

$$
\begin{aligned}
&A_m^{u_G(\hat{k})}(S_j, k) \\
&\quad = \mathrm{E}_{u_G(\hat{k})}\left\{ \sum_{i\in S_j}\sum_{l=1}^{\infty} e^{-\alpha v_{i,m,l}}\Bigg(\sum_{t=0}^{T_{i,m,l}(S_j^c)-1} \int_t^{t+1} \exp\{-\alpha \bar{I}_m^{-1}(s; v_{i,m,l})\} \right. \\
&\qquad\qquad \left. \times \exp[\alpha\{\bar{I}_m^{-1}(s; v_{i,m,l}) - s\}]\, ds\Bigg) \;\Bigg|\; \hat{k}(m)\right\} \\
&\quad \leq \mathrm{E}_{u_G(\hat{k})}\left\{ \sum_{i\in S_j}\sum_{l=1}^{\infty} e^{-\alpha v_{i,m,l}}\left[\sum_{t=0}^{T_{i,m,l}(S_j^c)-1} \int_t^{t+1} \exp\{-\alpha \bar{I}_m^{-1}(s; v_{i,m,l})\}\, ds\right] \;\Bigg|\; \hat{k}(m)\right\} \\
&\qquad + \alpha\, \mathrm{E}_{u_G(\hat{k})}\left[\sum_{i\in S_j}\sum_{l=1}^{\infty} e^{-\alpha v_{i,m,l}}\{\bar{I}_m^{-1}(T_{i,m,l}(S_j^c); v_{i,m,l}) - T_{i,m,l}(S_j^c)\} \right. \\
&\qquad\qquad \left. \times \left(\int_0^{T_{i,m,l}(S_j^c)} e^{-\alpha s}\, ds\right) \;\Bigg|\; \hat{k}(m)\right]. \qquad (29)
\end{aligned}
$$

Now inspect the first term on the right-hand side of (29). We observe that the first decision epoch at which machine m processes a job in S_j under control $u_G(\hat{k})$ is $\bar{I}_m^{-1}\{T_{\hat{k}(m)}(S_j^c)\}$ since the priority $S_j^c \to S_j$ is enforced. The first term gives a (discounted) measure of the availability of machine m from that point. Utilising this and the simple bound for the second term, we deduce from (29) that

$$
\begin{aligned}
A_m^{u_G(\hat{k})}(S_j, k) &\leq \mathrm{E}\left(\left[\int_{\bar{I}_m^{-1}\{T_{\hat{k}(m)}(S_j^c)\}}^{\infty} I_m(s)\, e^{-\alpha s}\, ds\right]\right) \\
&\quad + \alpha\, \mathrm{E}_{u_G(\hat{k})}\left[\sum_{i\in S_j}\sum_{l=1}^{\infty} e^{-\alpha v_{i,m,l}} T_{i,m,l}(S_j^c)\{\bar{I}_m^{-1}(T_{i,m,l}(S_j^c); v_{i,m,l}) - T_{i,m,l}(S_j^c)\} \;\Big|\; \hat{k}(m)\right]. \qquad (30)
\end{aligned}
$$

We now invoke Condition 1, together with the fact that

$$\mathrm{E}\{\bar{I}_m^{-1}(s;t) \mid \mathcal{F}_t\} \leq \mathrm{E}\{\bar{J}_M^{-1}(s;t) \mid \mathcal{F}_t\}$$

for all choices of s and t to bound the second term on the right-hand side of (30). This yields via a simple conditioning argument the inequality

$$
\begin{aligned}
&A_m^{u_G(\hat{k})}(S_j, k) \\
&\quad \leq \mathrm{E}\left(\left[\int_{\bar{I}_m^{-1}\{T_{\hat{k}(m)}(S_j^c)\}}^{\infty} I_m(s)\, e^{-\alpha s}\, ds\right]\right) \\
&\qquad + \alpha \max_{i\in S_j}[a\, \mathrm{E}(T_i(S_j^c)) + (b-1)\, \mathrm{E}\{(T_i(S_j^c))^2\}]\, \mathrm{E}_{u_G(\hat{k})}\left\{ \sum_{i\in S_j}\sum_{l=1}^{\infty} e^{-\alpha v_{i,m,l}} \;\Big|\; \hat{k}(m)\right\} \qquad (31) \\
&\quad \leq \{1 + K_1(S_j)\}\, \mathrm{E}\left(\left[\int_{\bar{I}_m^{-1}\{T_{\hat{k}(m)}(S_j^c)\}}^{\infty} I_m(s)\, e^{-\alpha s}\, ds\right]\right). \qquad (32)
\end{aligned}
$$

Note that (32) proceeds from (31) by bounding the second term in the latter via the inclusion

$$\{\nu_{i,m,l};\ i \in S_j,\ l \in \mathbb{Z}^+\} \subseteq \{t;\ t \in \mathbb{Z}^+,\ I_{m,t} = 1 \text{ and } t \geq \bar{I}_m^{-1}[T_{\hat{\boldsymbol{k}}(m)}(S_j^c)]\}.$$

The result now follows simply from inequality (32) and the identity

$$A^{u_G(\hat{\boldsymbol{k}})}(S_j, \boldsymbol{k}) = \sum_{m=1}^{M} A_m^{u_G(\hat{\boldsymbol{k}})}(S_j, \boldsymbol{k}). \tag{33}$$

This concludes the proof.

An improved upper bound is available under the condition

$$m \in M_t \quad \Longrightarrow \quad m \in M_s \qquad \forall\, s \in [t, \infty),\ 1 \leq m \leq M. \tag{34}$$

Condition (34) describes the situation of a 'machine arrival process' in which, once a machine is available, it remains permanently available thereafter. Note that this is a particular instance of Example 4 in Section 3 and includes the important special case $|M_t| = M,\ t \in \mathbb{N}$. Under this additional condition, the upper bound in Lemma 6 may be tightened as follows:

Lemma 7. *For Model 1, with the machine availability process satisfying (34),*

$$A^{u_G(\hat{\boldsymbol{k}})}(S_j, \boldsymbol{k}) \leq \mathrm{E}\Bigg(\sum_{m=1}^{M}\Bigg[\int_{\bar{I}_m^{-1}\{T_{\hat{\boldsymbol{k}}(m)}(S_j^c)\}}^{\infty} I_m(s)\, \mathrm{e}^{-\alpha s}\, \mathrm{d}s\Bigg]\Bigg), \qquad 1 \leq j \leq |E|,$$

for all initial states $\boldsymbol{k}$ *and associated partitions* $\hat{\boldsymbol{k}}$.

Proof. Follow the calculation of the proof of Lemma 6 through to (30) and note that, under condition (34)

$$\bar{I}_m^{-1}(T_{i,m,l}(S_j^c);\ \nu_{i,m,l}) = T_{i,m,l}(S_j^c).$$

Hence the second term on the right-hand side of (30) is zero under (34). Now use (33) to obtain the result.

Having obtained lower bounds for the $A(S_j, \boldsymbol{k})$ in Lemmas 3 and 4 and upper bounds for $A^u(S_j, \boldsymbol{k})$ in Lemmas 6 and 7, we are now in a position to use Corollary 1(ii) to bound the degree of reward suboptimality in the index-based policy $u_G(\hat{\boldsymbol{k}})$. In Theorem 1, we shall require the constants

$$K_2(S_j) = \frac{\alpha\, \mathrm{e}^{\alpha}}{2} \max_{i \in S_j} \mathrm{E}\{(T_i(S_j^c))^2\}, \qquad 1 \leq j \leq |E|,$$

$$K_3(S_j, \hat{\boldsymbol{k}}) = \alpha \sum_{m=1}^{M} (a\, \mathrm{E}\{T_{\hat{\boldsymbol{k}}(m)}(S_j^c)\} + b\, \mathrm{E}[\{T_{\hat{\boldsymbol{k}}}(S_j^c)\}^2]), \qquad 1 \leq j \leq |E|,$$

in addition to the $K_1(S_j)$, $1 \leq j \leq |E|$, given in (27).

Theorem 1. (Closeness to optimality of index-based heuristic: Model 1.) (i) *When $\mathcal{U}$ is the class of all non-anticipative controls for Model 1 and the machine availability process satisfies Condition 1,*

$$\begin{aligned} R^{\mathrm{opt}}(\boldsymbol{k}) - R^{u_G(\hat{\boldsymbol{k}})}(\boldsymbol{k}) &\leq \sum_{j=1}^{|E|-1} (G_{j+1} - G_j)\left[\{K_1(S_j) + K_2(S_j)\}\,\mathrm{E}\left\{\int_0^\infty M(s)\,\mathrm{e}^{-\alpha s}\,\mathrm{d}s\right\} + K_3(S_j, \hat{\boldsymbol{k}})\right] \\ &= O(1); \end{aligned} \tag{35}$$

(ii) *When $\mathcal{U}$ is the class of all non-anticipative local controls for Model 1 and the machine availability process satisfies Condition 1,*

$$\begin{aligned} R^{\mathrm{opt}}(\boldsymbol{k}) - R^{u_G(\hat{\boldsymbol{k}})}(\boldsymbol{k}) &\leq \sum_{j=1}^{|E|-1} (G_{j+1} - G_j)\left[K_1(S_j)\,\mathrm{E}\left\{\int_0^\infty M(s)\,\mathrm{e}^{-\alpha s}\,\mathrm{d}s\right\} + K_3(S_j, \hat{\boldsymbol{k}})\right] \\ &= O(1); \end{aligned} \tag{36}$$

(iii) *When $\mathcal{U}$ is the class of all non-anticipative local controls for Model 1 and the machine availability process satisfies (34) and Condition 1′,*

$$R^{\mathrm{opt}}(\boldsymbol{k}) - R^{u_G(\hat{\boldsymbol{k}})}(\boldsymbol{k}) \leq \sum_{j=1}^{|E|-1} (G_{j+1} - G_j) K_3(S_j, \hat{\boldsymbol{k}}) = O(\alpha). \tag{37}$$

These results hold for all initial states $\boldsymbol{k}$ and associated partitions $\hat{\boldsymbol{k}}$.

Proof. (i) From Corollary 1(ii) we have that

$$\begin{aligned} R^{\mathrm{opt}}(\boldsymbol{k}) - R^{u_G(\hat{\boldsymbol{k}})}(\boldsymbol{k}) &\leq \sum_{j=1}^{|E|-1} (G_{j+1} - G_j)\{A^{u_G(\hat{\boldsymbol{k}})}(S_j, \boldsymbol{k}) - A(S_j, \boldsymbol{k})\} \\ &\leq \sum_{j=1}^{|E|-1} (G_{j+1} - G_j)\left[\mathrm{E}\left(\sum_{m=1}^{M}\left[\int_{\bar{I}_m^{-1}\{T_{\hat{\boldsymbol{k}}(m)}(S_j^{\mathrm{c}})\}}^{\infty} I_m(s)\,\mathrm{e}^{-\alpha s}\,\mathrm{d}s\right]\right)\right. \\ &\qquad \left. - \mathrm{E}\left\{\int_{\bar{M}^{-1}(T_{\boldsymbol{k}}(S_j^{\mathrm{c}}))}^{\infty} M(s)\,\mathrm{e}^{-\alpha s}\,\mathrm{d}s\right\}\right] \\ &\quad + \sum_{j=1}^{|E|-1} (G_{j+1} - G_j)\left[K_1(S_j)\,\mathrm{E}\left(\sum_{m=1}^{M}\left[\int_{\bar{I}_m^{-1}\{T_{\hat{\boldsymbol{k}}(m)}(S_j^{\mathrm{c}})\}}^{\infty} I_m(s)\,\mathrm{e}^{-\alpha s}\,\mathrm{d}s\right]\right)\right. \\ &\qquad \left. + K_2(S_j)\,\mathrm{E}\left\{\int_{\bar{M}^{-1}(T_{\boldsymbol{k}}(S_j^{\mathrm{c}}))}^{\infty} M(s)\,\mathrm{e}^{-\alpha s}\,\mathrm{d}s\right\}\right], \end{aligned} \tag{38}$$

where the second inequality, (38), utilises Lemmas 3 and 6. We now analyse the right-hand side of (38). Utilising the facts that $1 \geq \mathrm{e}^{-\alpha t} \geq 1 - \alpha t$, $\alpha \geq 0$, $t \geq 0$, and $M(t) = \sum_{m=1}^{M} I_m(t)$,

$t \geq 0$, we deduce that

$$\begin{aligned}
&\mathrm{E}\Bigg(\sum_{m=1}^{M}\Bigg[\int_{\bar{I}_m^{-1}\{T_{\hat{k}(m)}(S_j^{\mathrm{c}})\}}^{\infty} I_m(s)\,\mathrm{e}^{-\alpha s}\,\mathrm{d}s\Bigg]\Bigg) - \mathrm{E}\left\{\int_{\bar{M}^{-1}(T_{\boldsymbol{k}}(S_j^{\mathrm{c}}))}^{\infty} M(s)\,\mathrm{e}^{-\alpha s}\,\mathrm{d}s\right\} \\
&\quad = \mathrm{E}\left\{\int_0^{\bar{M}^{-1}(T_{\boldsymbol{k}}(S_j^{\mathrm{c}}))} M(s)\,\mathrm{e}^{-\alpha s}\,\mathrm{d}s\right\} - \mathrm{E}\Bigg(\sum_{m=1}^{M}\Bigg[\int_0^{\bar{I}_m^{-1}\{T_{\hat{k}(m)}(S_j^{\mathrm{c}})\}} I_m(s)\,\mathrm{e}^{-\alpha s}\,\mathrm{d}s\Bigg]\Bigg) \\
&\quad \leq \mathrm{E}\left\{\int_0^{\bar{M}^{-1}(T_{\boldsymbol{k}}(S_j^{\mathrm{c}}))} M(s)\,\mathrm{d}s\right\} - \mathrm{E}\Bigg(\sum_{m=1}^{M}\Bigg[\int_0^{\bar{I}_m^{-1}\{T_{\hat{k}(m)}(S_j^{\mathrm{c}})\}} I_m(s)\,\mathrm{d}s\Bigg]\Bigg) \\
&\qquad + \alpha\,\mathrm{E}\Bigg(\sum_{m=1}^{M}\Bigg[\int_0^{\bar{I}_m^{-1}\{T_{\hat{k}(m)}(S_j^{\mathrm{c}})\}} s\,I_m(s)\,\mathrm{d}s\Bigg]\Bigg) \\
&\quad \leq \mathrm{E}\left[\left\{T_{\boldsymbol{k}}(S_j^{\mathrm{c}}) - \sum_{m=1}^{M} T_{\hat{k}(m)}(S_j^{\mathrm{c}})\right\}\right] + \alpha\,\mathrm{E}\Bigg(\sum_{m=1}^{M}[T_{\hat{k}(m)}(S_j^{\mathrm{c}})\bar{I}_m^{-1}\{T_{\hat{k}(m)}(S_j^{\mathrm{c}})\}]\Bigg) \qquad (39) \\
&\quad \leq K_3(S_j, \hat{\boldsymbol{k}}), \qquad 1 \leq j \leq |E| - 1, \qquad (40)
\end{aligned}$$

by application of Condition 1 to the second term in (39). Note that the first term is zero. We also observe that

$$\begin{aligned}
&\max\left\{\mathrm{E}\Bigg(\sum_{m=1}^{M}\Bigg[\int_{\bar{I}_m^{-1}\{T_{\hat{k}(m)}(S_j^{\mathrm{c}})\}}^{\infty} I_m(s)\,\mathrm{e}^{-\alpha s}\,\mathrm{d}s\Bigg]\Bigg), \mathrm{E}\left\{\int_{\bar{M}^{-1}(T_{\boldsymbol{k}}(S_j^{\mathrm{c}}))}^{\infty} M(s)\,\mathrm{e}^{-\alpha s}\,\mathrm{d}s\right\}\right\} \\
&\quad \leq \mathrm{E}\left\{\int_0^{\infty} M(s)\,\mathrm{e}^{-\alpha s}\,\mathrm{d}s\right\}. \qquad (41)
\end{aligned}$$

We now obtain the inequality in (35) by combining inequalities (38), (40) and (41).

Consider now the behaviour of the right-hand side of (35) in the limit $\alpha \to 0$. Note that it follows that, since all positive entries in the vector $\boldsymbol{A}^S$ are $O(1)$ for all subsets $S \subseteq E$, then so are all indices G_j, $1 \leq j \leq |E|$. Observe that the constants $K_1(S_j)$, $K_2(S_j)$ and $K_3(S_j, \hat{\boldsymbol{k}})$ are all $O(\alpha)$. It then follows from the fact that

$$\mathrm{E}\left\{\int_0^{\infty} M(s)\,\mathrm{e}^{-\alpha s}\,\mathrm{d}s\right\} = O(1/\alpha)$$

and is in fact bounded by M/α, that the bound in (35) is $O(1)$.

(ii) As above, but use Lemmas 4 and 6 for the calculation.

(iii) As above, but use Lemmas 4 and 7 for the calculation.

Remarks. 1. Note that model assumptions guarantee that the expected rewards $R^{\mathrm{opt}}(\boldsymbol{k})$ and $R^{u_G(\hat{\boldsymbol{k}})}(\boldsymbol{k})$ analysed in Theorem 1 are both $O(1/\alpha)$ while the suboptimality bounds are proved to be $O(1)$ or $O(\alpha)$. These results yield various forms of asymptotic optimality for a suitably defined limiting form of the policy $u_G(\hat{\boldsymbol{k}})$ as $\alpha \to 0$. This will be explained further in Section 7.

2. Theorem 1 holds for all partitions $\hat{\boldsymbol{k}}$. To guide a choice of partition, it would seem sensible to aim to produce as small a bound as possible in whichever is appropriate of (35), (36) and

(37). Since K_3 is the only constant which depends on the partition, by inspection of Theorem 1 we recommend a choice of $\hat{\boldsymbol{k}}$ which minimises

$$\sum_{j=1}^{|E|-1}\sum_{m=1}^{M}(G_{j+1}-G_j)(a\,\mathrm{E}\{T_{\hat{\boldsymbol{k}}(m)}(S_j^{\mathrm{c}})\}+b\,\mathrm{E}[\{T_{\hat{\boldsymbol{k}}(m)}(S_j^{\mathrm{c}})\}^2]).$$

This minimisation can be thought of as a load balancing problem. Its solution in simple cases can be seen to encourage an equal distribution of the work in the system among the machines, as would seem sensible.

6. On the evaluation of a class of index-based controls for discounted multi-armed bandits

We now consider Model 2. Here we use $\boldsymbol{k} \in \times_b E_b$ for the initial state of the system, namely the B-vector of states of the individual bandits, at time $t = 0$. Recall that the members of $E = \bigcup_b E_b$ are numbered in decreasing order of their Gittins index such that

$$G_{|E|} \geq G_{|E|-1} \geq \cdots \geq G_2 \geq G_1. \tag{42}$$

We use $\underline{b}$ to label the state of lowest index for bandit b, given by

$$\underline{b} = \min\{i;\ i \in E_b\}$$

and then number the B bandits such that

$$\underline{1} \geq \underline{2} \geq \cdots \geq \underline{B-1} \geq \underline{B}. \tag{43}$$

We design a control u_G for the multi-armed bandit to be an index-based policy which is structured as follows:

(a) for $b = 1, 2, \ldots, M-1$, bandit b is chosen for processing at time $t \in \mathbb{N}$ if and only if $|M_t| \geq b$;

(b) the bandits numbered $M, M+1, \ldots, B$ are only chosen for processing when $|M_t| = M$. At such epochs, a bandit is scheduled from among this collection which has the largest Gittins index. Denote the initial state of this collection by $\boldsymbol{k}_M$.

Hence according to u_G, bandit 1 with the largest *guaranteed* reward rate (i.e. the largest minimal index) is scheduled as a first priority on the machines available at any time, then bandit 2 and so on. The collection of bandits remaining from these allocations for processing *only* when all M machines are available are subject to a Gittins index policy for selection. Recall the definitions of the random variables $T_{\boldsymbol{k}}(S_j^{\mathrm{c}})$ from Section 4.

Lemma 8. *For Model 2, with the machine availability process satisfying Condition 1,*

$$A^{u_G}(S_j, \boldsymbol{k}) \leq \{1 + K_1(S_j)\}\,\mathrm{E}\left[\sum_{m=\mu_j+1}^{M}\left\{\int_{\bar{J}_m^{-1}(T_{k_m}(S_j^{\mathrm{c}}))}^{\infty} J_m(s)\,\mathrm{e}^{-\alpha s}\,\mathrm{d}s\right\}\right], \quad 0 \leq \mu_j \leq M-1; \tag{44}$$

and

$$A^{u_G}(S_j, \boldsymbol{k}) = 0, \qquad \mu_j \geq M, \tag{45}$$

for all initial states $\boldsymbol{k}$*, where the constants* $K_1(S_j)$ *are given by the expression in (27).*

Proof. Under control u_G, no machine ever schedules a member of E drawn from subset $S_{\underline{M}-1} = \{\underline{M}-1, \underline{M}-2, \ldots, 1\}$. By definition, this is the collection of j for which $\mu_j \geq M$ and (45) above follows.

Suppose now that $1 \leq m \leq M-1$ and let $A_m^{u_G}(S_j, \boldsymbol{k})$ denote the contribution to $A^{u_G}(S_j, \boldsymbol{k})$ from the processing of bandit m under u_G at those times $t \in \mathbb{N}$ for which $|M_t| \geq m$. Firstly observe that, from the definitions of the quantities involved,

$$j \leq \underline{m} - 1 \quad \Longleftrightarrow \quad \mu_j \geq m \tag{46}$$

and that both conditions in (46) imply that $A_m^{u_G}(S_j, \boldsymbol{k})$ is zero. Now suppose that $m \geq \mu_j + 1$. As in (28) we have that

$$A_m^{u_G}(S_j, \boldsymbol{k}) = \mathrm{E}_{u_G}\left(\sum_{i \in S_j} \sum_{l=1}^{\infty} \mathrm{e}^{-\alpha v_{i,m,l}} \int_0^{T_{i,m,l}(S_j^{\mathrm{c}})} \mathrm{e}^{-\alpha s}\, \mathrm{d}s \;\middle|\; k_m \right), \tag{47}$$

where now $v_{i,m,l}$ stands for the lth occasion upon which, during the processing of bandit m, state $i \in S_j$ is encountered and k_m is its initial state. We now re-write (47) and utilise the facts that $\mathrm{e}^{\alpha t} \leq 1 + \alpha t\, \mathrm{e}^{\alpha t}$, $\alpha \geq 0$, $t \geq 0$, and that $\bar{J}_m^{-1}(s; t) - s$ is non-decreasing in s for all $t \geq 0$ to obtain that

$$\begin{aligned}
&A_m^{u_G}(S_j, \boldsymbol{k}) \\
&\quad = \mathrm{E}_{u_G}\left\{ \sum_{i \in S_j} \sum_{l=1}^{\infty} \mathrm{e}^{-\alpha v_{i,m,l}} \Bigg(\sum_{t=0}^{T_{i,m,l}(S_j^{\mathrm{c}})-1} \int_t^{t+1} \exp\{-\alpha \bar{J}_m^{-1}(s; v_{i,m,l})\} \right. \\
&\qquad\qquad \left. \times \exp[\alpha\{\bar{J}_m^{-1}(s; v_{i,m,l}) - s\}]\, \mathrm{d}s \Bigg) \;\middle|\; k_m \right\} \\
&\quad \leq \mathrm{E}\left\{ \int_{\bar{J}_m^{-1}(T_{k_m}(S_j^{\mathrm{c}}))}^{\infty} J_m(t)\, \mathrm{e}^{-\alpha t}\, \mathrm{d}t \right\} \\
&\qquad + \alpha\, \mathrm{E}_{u_G}\left[\sum_{i \in S_j} \sum_{l=1}^{\infty} \mathrm{e}^{-\alpha v_{i,m,l}} T_{i,m,l}(S_j^{\mathrm{c}}) \{\bar{J}_m^{-1}(T_{i,m,l}(S_j^{\mathrm{c}}); v_{i,m,l}) - T_{i,m,l}(S_j^{\mathrm{c}})\} \;\middle|\; k_m \right].
\end{aligned}$$

The details are similar to those which yield (30) in the proof of Lemma 6. Invoking Condition 1, together with the fact that

$$\mathrm{E}\{\bar{J}_m^{-1}(s; t) \mid \mathcal{F}_t\} \leq \mathrm{E}\{\bar{J}_M^{-1}(s; t) \mid \mathcal{F}_t\}$$

for all choices of s and t yields

$$A_m^{u_G}(S_j, \boldsymbol{k}) \leq \{1 + K_1(S_j)\}\, \mathrm{E}\left\{ \int_{\bar{J}_m^{-1}(T_{k_m}(S_j^{\mathrm{c}}))}^{\infty} J_m(s)\, \mathrm{e}^{-\alpha s}\, \mathrm{d}s \right\}. \tag{48}$$

The details are again similar to those in the proof of Lemma 6 and are omitted.

We now write $A_M^{u_G}(S_j, \boldsymbol{k})$ for the contribution to $A^{u_G}(S_j, \boldsymbol{k})$ from the processing of the bandits $M, M+1, \ldots, B$ at those epochs for which $|M_t| = M$. By a similar account to the above, we have that

$$A_M^{u_G}(S_j, \boldsymbol{k}) \leq \{1 + K_1(S_j)\}\, \mathrm{E}\left\{ \int_{\bar{J}_M^{-1}(T_{k_M}(S_j^{\mathrm{c}}))}^{\infty} J_M(s)\, \mathrm{e}^{-\alpha s}\, \mathrm{d}s \right\}. \tag{49}$$

Finally, observing that when $0 \leq \mu_j \leq M-1$,

$$A^{u_G}(S_j, \boldsymbol{k}) = \sum_{m=\mu_j+1}^{M} A_m^{u_G}(S_j, \boldsymbol{k}), \tag{50}$$

(44) now follows from (48)–(50). This concludes the proof.

The proof of Lemma 9 modifies that of Lemma 8 in much the same way that the proof of Lemma 7 modifies that of Lemma 6. It is omitted.

Lemma 9. *For Model 2, with $|M_t|$ non-decreasing in t, we have that*

$$A^{u_G}(S_j, \boldsymbol{k}) \leq \mathrm{E}\Bigg[\sum_{m=\mu_j+1}^{M} \left\{ \int_{\bar{J}_m^{-1}(T_{k_m}(S_j^c))}^{\infty} J_m(s)\, \mathrm{e}^{-\alpha s}\, \mathrm{d}s \right\}\Bigg], \qquad 0 \leq \mu_j \leq M-1;$$

and

$$A^{u_G}(S_j, \boldsymbol{k}) = 0, \qquad \mu_j \geq M$$

for all initial states $\boldsymbol{k}$.

We now use Corollary 1(ii) to bound the degree of reward suboptimality in the index-based policy u_G. Theorem 2 follows from Corollary 1(ii) together with Lemmas 5, 8 and 9 through an analysis similar to that in the proof of Theorem 1. The statement of Theorem 2 requires the constants

$$K_4(S_j) = \alpha \sum_{m=\mu_j+1}^{M} [a\,\mathrm{E}(T_{k_m}(S_j^c)) + b\,\mathrm{E}\{(T_{k_m}(S_j^c))^2\}]$$

for the range $0 \leq \mu_j \leq M-1$ with a and b as in (9), together with the constants $K_1(S_j)$, $1 \leq j \leq |E|$, given in (27). Its proof is omitted.

Theorem 2. (Closeness to optimality of index-based heuristic: Model 2.) (i) *When $\mathcal{U}$ is the class of all non-anticipative, non-idling controls for Model 2, with the machine availability process satisfying Condition 1,*

$$\begin{aligned} &R^{\mathrm{opt}}(\boldsymbol{k}) - R^{u_G}(\boldsymbol{k}) \\ &\quad \leq \sum_{j=1}^{|E|-1} (G_{j+1} - G_j)\left[K_1(S_j)\,\mathrm{E}\left\{\int_0^{\infty} M(s)\,\mathrm{e}^{-\alpha s}\,\mathrm{d}s\right\} + K_4(S_j)\right] I(0 \leq \mu_j \leq M-1) \\ &\quad = O(1); \end{aligned}$$

(ii) *When $\mathcal{U}$ is the class of all non-anticipative, non-idling controls for Model 2, with $|M_t|$ non-decreasing in t almost surely,*

$$R^{\mathrm{opt}}(\boldsymbol{k}) - R^{u_G}(\boldsymbol{k}) \leq \sum_{j=1}^{|E|-1} (G_{j+1} - G_j) K_4(S_j) I(0 \leq \mu_j \leq M-1) = O(\alpha).$$

These results hold for all initial states $\boldsymbol{k}$.

Remarks. 1. The first comment following Theorem 1 applies here to the control u_G.

2. A simpler analysis is available for a variant of Model 2 in which non-availability occurs at decision epochs when a project being processed and the machine upon which it is being processed crash together. We model such a phenomenon by extending each bandit state space

to $E_b \cup \{0_b\} \equiv \tilde{E}_b$, where 0_b is a 'crash' state. The appropriate irreducible transition law $\tilde{P}^b$ is over $\tilde{E}_b$, $1 \le b \le B$. No rewards are earned in crash states. With this modelling approach, we can now take $|M_t| = M$, $t \in \mathbb{N}$, with those machines processing bandits in crash states being essentially unavailable. The class of admissible controls needs to *impose* the priority $\bigcup_{b=1}^{B} \{0_b\} \to \bigcup_{b=1}^{B} E_b \equiv E$ to guarantee this. This imposition necessitates the development of new random variables $\tilde{T}_i(S^c)$, $i \in S \cap E_b$, $1 \le b \le B$ for $S \subseteq E$, as follows: consider a set-up in which only bandit b in state i is present at time 0. Under processing by a single machine ($|M_t| = 1$, $t \in \mathbb{N}$), bandit b evolves under transition law $\tilde{P}^b$. In particular, it may spend time in crash state 0_b. We write $\tilde{T}_i(S^c)$ for the first time at or after time 1 at which bandit b enters set S. From these random variables we develop $\tilde{\boldsymbol{A}} = \{\tilde{\boldsymbol{A}}^S, S \subseteq E\}$ and Gittins indices $\tilde{G}_j$, $j \in E$, as in Section 4. The sets $\tilde{S}_j$, and the values $\tilde{\mu}_j$, $1 \le j \le |E|$, are defined with respect to these indices. From the indices we develop a numbering of the bandits as in (42) and (43).

The control $u_{\tilde{G}}$ of interest is an index-based policy structured as follows:

(a) for $b = 1, 2, \ldots, M-1$, machine b processes bandit b at all decision epochs;

(b) machine M processes bandits $M, M+1, \ldots, B$ according to a Gittins index policy. Such a policy is understood to impose the priority $\bigcup_{b=M}^{B} \{0_b\} \to \bigcup_{b=M}^{B} E_b$ and to choose from the latter set according to index values.

In evaluating control $u_{\tilde{G}}$, we observe that since we can assume $|M_t| = M$, $t \in \mathbb{N}$, then Condition 1 is trivially satisfied with $a = 0$, $b = 1$. The $O(\alpha)$ suboptimality bound in Theorem 3 follows.

Theorem 3. (Closeness to optimality of index-based heuristic: 'crash' model.) *When $\mathcal{U}$ is the class of all non-anticipative, non-idling controls for the above model which impose the priority $\bigcup_{b=1}^{B} \{0_b\} \to E$,*

$$R^{\mathrm{opt}}(\boldsymbol{k}) - R^{u_{\tilde{G}}}(\boldsymbol{k}) \le \sum_{j=1}^{|E|-1} (\tilde{G}_{j+1} - \tilde{G}_j) \tilde{K}_4(\tilde{S}_j) I(0 \le \tilde{\mu}_j \le M-1) = O(\alpha),$$

for all initial states $\boldsymbol{k}$, where

$$\tilde{K}_4(\tilde{S}_j) = \alpha \sum_{m=\tilde{\mu}_j+1}^{M} \mathrm{E}\{(T_{k_m}(\tilde{S}_j^c))^2\}.$$

7. Asymptotic optimality

In Theorems 1–3 we have obtained bounds on the degree of reward suboptimality of index-based controls. Some of these bounds are $O(1)$ and some $O(\alpha)$. We now take the further natural step in the analysis and explore what these results imply when we take the limit $\alpha \to 0$. To facilitate the discussion we include α in the notations $u_G(\hat{\boldsymbol{k}}, \alpha)$, $u_G(\alpha)$, $R^u(\boldsymbol{k}, \alpha)$, $R^{\mathrm{opt}}(\boldsymbol{k}, \alpha)$, $\boldsymbol{A}^S(\alpha)$ and $G_j(\alpha)$.

It is trivial to establish, from the fact that in all cases $\lim_{\alpha \to 0} A_i^S(\alpha) = \mathrm{E}(T_i(S^c))$, $i \in S$, and from the structure of the adaptive greedy algorithm, that the limits

$$\lim_{\alpha \to 0} G_j(\alpha) \equiv \bar{G}_j, \qquad 1 \le j \le |E|,$$

all exist and are finite. Hence for both Models 1 and 2 the index-based heuristics studied in Sections 5 and 6 respectively have associated limit policies $u_{\bar{G}}(\hat{\boldsymbol{k}})$ and $u_{\bar{G}}$. These share the structure of $u_G(\hat{\boldsymbol{k}}, \alpha)$ and $u_G(\alpha)$ respectively but utilise an ordering on the set E given by the limiting indices $\bar{G}_j$, $j \in E$.

In Theorems 4–7, various forms of asymptotic optimality are claimed for our limit policies $u_{\bar{G}}(\hat{\boldsymbol{k}})$ and $u_{\bar{G}}$. A central notion is that of *n-discount optimality*, where we say that control u is *n-discount optimal* if

$$\lim_{\alpha \to 0} \alpha^{-n} \{R^{\mathrm{opt}}(\boldsymbol{k}, \alpha) - R^{u}(\boldsymbol{k}, \alpha)\} = 0.$$

The reader should consult Puterman (1994) for a full discussion of this and other notions of asymptotic optimality set in the context of Markov decision processes.

Theorem 4. (Asymptotic optimality of limit policy $u_{\bar{G}}(\hat{\boldsymbol{k}})$: Model 1.) (i) *When $\mathcal{U}$ is the class of all non-anticipative controls for Model 1 and the machine availability process satisfies Condition 1, then limit policy $u_{\bar{G}}(\hat{\boldsymbol{k}})$ is (-1)-discount optimal and is such that*

$$\lim_{\alpha \to 0} \left\{ \frac{R^{\mathrm{opt}}(\boldsymbol{k}, \alpha) - R^{u_{\bar{G}}(\hat{\boldsymbol{k}})}(\boldsymbol{k}, \alpha)}{R^{\mathrm{opt}}(\boldsymbol{k}, \alpha)} \right\} = 0$$

for all choices of $\hat{\boldsymbol{k}}$;

(ii) *When $\mathcal{U}$ is the class of all non-anticipative local controls for Model 1, the machine availability process satisfies (34) and Condition 1′ and the limiting indices are distinct then the limit policy $u_{\bar{G}}(\hat{\boldsymbol{k}})$ is 0-discount optimal for all choices of $\hat{\boldsymbol{k}}$.*

Sketch proof. From Corollary 1(ii) we have, in an obvious notation, that

$$R^{\mathrm{opt}}(\boldsymbol{k}, \alpha) - R^{u_{\bar{G}}(\hat{\boldsymbol{k}})}(\boldsymbol{k}, \alpha) \leq \sum_{j=1}^{|E|-1} \{G_{j+1}(\alpha) - G_j(\alpha)\} \{A^{u_{\bar{G}}(\hat{\boldsymbol{k}})}(S_j, \boldsymbol{k}, \alpha) - A(S_j, \boldsymbol{k}, \alpha)\}, \quad (51)$$

where the class numbering used in (51) is in general α-dependent. Write

$$E = \bigcup_{r=1}^{R} E_r,$$

where each set E_r is maximal, containing customer classes whose limiting indices are equal. Let $\bar{G}^{(r)}$ be the common value of $\bar{G}_j$ for $j \in E_r$. We suppose that

$$\bar{G}^{(R)} > \bar{G}^{(R-1)} > \cdots > \bar{G}^{(1)}.$$

There must exist $\bar{\alpha}$ such that for $\alpha \in (0, \bar{\alpha}]$ both $u_G(\hat{\boldsymbol{k}}, \alpha)$ and $u_{\bar{G}}(\hat{\boldsymbol{k}})$ enforce the priorities $E_R \to E_{R-1} \to \cdots \to E_1$. Let

$$\bar{S}_r = \bigcup_{s=1}^{r} E_s, \qquad 1 \leq r \leq R.$$

Utilising the fact that

$$G_k(\alpha) - G_l(\alpha) \leq O(\alpha), \qquad k, l \in E_r, \ 1 \leq r \leq R,$$

we can rewrite (51) as

$$R^{\mathrm{opt}}(\boldsymbol{k},\alpha) - R^{u_{\bar{G}}(\hat{\boldsymbol{k}})}(\boldsymbol{k},\alpha) \le \sum_{r=1}^{R-1} \{\bar{G}^{(r+1)} - \bar{G}^{(r)}\}\{A^{u_{\bar{G}}(\hat{\boldsymbol{k}})}(\bar{S}_r, \boldsymbol{k}, \alpha) - A(\bar{S}_r, \boldsymbol{k}, \alpha)\} + O(1). \tag{52}$$

But since both $u_G(\hat{\boldsymbol{k}}, \alpha)$ and $u_{\bar{G}}(\hat{\boldsymbol{k}})$ enforce the priorities $\bar{S}_r^{\mathrm{c}} \to \bar{S}_r$ for all r and $\alpha \in (0, \bar{\alpha}]$, we conclude from the calculations in Section 5 that

$$A^{u_{\bar{G}}(\hat{\boldsymbol{k}})}(\bar{S}_r, \boldsymbol{k}, \alpha) - A(\bar{S}_r, \boldsymbol{k}, \alpha) \le O(1), \qquad \alpha \in (0, \bar{\alpha}],\ 1 \le r \le R, \tag{53}$$

for all choices of partition $\hat{\boldsymbol{k}}$. From (52) and (53) it follows that

$$R^{\mathrm{opt}}(\boldsymbol{k},\alpha) - R^{u_{\bar{G}}(\hat{\boldsymbol{k}})}(\boldsymbol{k},\alpha) \le O(1), \qquad \alpha \in (0, \bar{\alpha}],$$

and the conclusions of Theorem 4(i) follow easily.

The hypotheses of (ii) guarantee that $u_{\bar{G}}(\hat{\boldsymbol{k}})$ coincides with $u_G(\hat{\boldsymbol{k}}, \alpha)$ for all $\alpha \in (0, \hat{\alpha}]$, for some $\hat{\alpha} > 0$. Theorem 4(ii) then follows easily from Theorem 1(iii).

The proof of Theorem 5 is along similar lines and is omitted.

Theorem 5. (Asymptotic optimality of limit policy $u_{\bar{G}}$: Model 2.) (i) *When $\mathcal{U}$ is the class of all non-anticipative, non-idling controls for Model 2, and the machine availability process satisfies Condition 1, then limit policy $u_{\bar{G}}$ is (-1)-discount optimal and is such that*

$$\lim_{\alpha \to 0} \left\{ \frac{R^{\mathrm{opt}}(\boldsymbol{k},\alpha) - R^{u_{\bar{G}}}(\boldsymbol{k},\alpha)}{R^{\mathrm{opt}}(\boldsymbol{k},\alpha)} \right\} = 0;$$

(ii) *When $\mathcal{U}$ is the class of all non-anticipative, non-idling controls for Model 2, with $|M_t|$ non-decreasing in t almost surely, and the limiting indices are distinct, then the limit policy $u_{\bar{G}}$ is 0-discount optimal.*

We continue our consideration of Model 2 by imposing the additional condition that the process $\{|M_t|, t \in \mathbb{N}\}$ be an irreducible Markov chain, as in Example 1. The multi-armed bandit of Model 2 can then be viewed as a (finite state, finite action) Markov decision process and we can utilise the classical theory of Blackwell (1962), Veinott (1966) and Denardo and Miller (1968). We firstly observe that straightforward analysis yields the existence of a sequence $\{\alpha_n,\ n \in \mathbb{N}\}$ with $\lim_{n\to\infty} \alpha_n = 0$ together with a numbering of members of E for which

(i) $G_{|E|}(\alpha_n) \ge G_{|E|-1}(\alpha_n) \ge \cdots \ge G_2(\alpha_n) \ge G_1(\alpha_n),\ n \in \mathbb{N}$;

(ii) $\bar{G}_{|E|} \ge \bar{G}_{|E|-1} \ge \cdots \ge \bar{G}_2 \ge \bar{G}_1$

in all cases. The limit policies discussed in Theorems 6 and 7 are constructed via the index ordering $|E| \to |E-1| \to \cdots \to 2 \to 1$ in (i) and (ii) above. These results follow from Theorems 2(i) and 3 respectively by the use of arguments which are standard in this area. The proofs are omitted.

Theorem 6. (Asymptotic optimality of limit policy $u_{\bar{G}}$: Markovian Model 2.) *When $\mathcal{U}$ is the class of all non-anticipative, non-idling controls for Model 2 and process $\{|M_t|,\ t \in \mathbb{N}\}$ is an irreducible Markov chain, then limit policy $u_{\bar{G}}$ is average-reward optimal.*

We shall use $\bar{u}_{\tilde{G}}$ to denote the appropriate limit policy for the 'crash' model described in the second remark at the end of Section 6.

Theorem 7. (Asymptotic optimality of limit policy $\bar{u}_{\tilde{G}}$: 'crash' model.) *When $\mathcal{U}$ is the class of all non-anticipative, non-idling controls for the 'crash' model, which impose the priority $\bigcup_{b=1}^{B}\{0_b\} \rightarrow E$, then the limit policy $\bar{u}_{\tilde{G}}$ is 0-discount optimal and average-overtaking optimal.*

Acknowledgements

This work is supported by the Engineering and Physical Sciences Research Council both through the award of grant GR/M09308 and via a research studentship for the first author.

References

BERTSIMAS, D. AND NIÑO-MORA, J. (1996). Conservation laws, extended polymatroids and multi-armed bandit problems: a polyhedral approach to indexable systems. *Math. Operat. Res.* **21,** 257–306.

BIRGE, J., FRENK, J. B. G., MITTENTHAL, J. AND RINNOOY KAN, A. H. G. (1990). Single machine scheduling subject to stochastic breakdowns. *Naval Res. Logist.* **37,** 660–677.

BLACKWELL, D. (1962). Discrete dynamic programming. *Ann. Math. Statist.* **33,** 719–726.

DACRE, M., GLAZEBROOK, K. D. AND NIÑO-MORA, J. (1999). The achievable region approach to the optimal control of stochastic systems (with discussion). *J. R. Statist. Soc. B* **61,** 747–791.

DENARDO, E. V. AND MILLER, B. L. (1968). An optimality criterion for discrete dynamic programming with no discounting. *Ann. Math. Statist.* **39,** 1220–1227.

GITTINS, J. C. AND JONES, D. M. (1974). A dynamic allocation index for the sequential design of experiments. In *Progress in Statistics* (European Meeting of Statisticians, Budapest, 1972), eds J. Gani, K. Sarkadi and I. Vince. North-Holland, Amsterdam, pp. 241–266.

GLAZEBROOK, K. D. (1976). Stochastic scheduling with order constraints. *Int. J. Systems Sci.* **7,** 657–666.

GLAZEBROOK, K. D. (1984). Scheduling stochastic jobs on a single machine subject to breakdowns. *Naval Res. Logist. Quart.* **31,** 251–264.

GLAZEBROOK, K. D. (1987). Evaluating the effects of machine breakdowns in stochastic scheduling problems. *Naval Res. Logist.* **34,** 319–335.

GLAZEBROOK, K. D. AND GARBE, R. (1999). Almost optimal policies for stochastic systems which almost satisfy conservation laws. *Ann. Operat. Res.* **92,** 19–43.

GLAZEBROOK, K. D. AND NIÑO-MORA, J. (2001). Scheduling multiclass queueing networks on parallel servers: approximate and heavy-traffic optimality of Klimov's rule. To appear in *Operat. Res.*

GLAZEBROOK, K. D. AND WILKINSON, D. J. (2000). Index-based policies for discounted multi-armed bandits on parallel machines. *Ann. Appl. Prob.* **10,** 877–896.

KATEHAKIS, M. N. AND VEINOTT, A. F. (1987). The multi-armed bandit problem: decomposition and computation. *Math. Operat. Res.* **12,** 262–268.

PINEDO, M. AND RAMMOUZ, E. (1988). A note on stochastic scheduling on a single machine subject to breakdown and repair. *Prob. Eng. Inf. Sci.* **2,** 41–49.

PUTERMAN, M. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley, New York.

VEINOTT, A. F. JR. (1966). On finding optimal policies in discrete dynamic programming with no discounting. *Ann. Math. Statist.* **37,** 1284–1294.

WEBER, R. R. (1982). Scheduling jobs with stochastic processing requirements on parallel machines to minimize makespan or flowtime. *J. Appl. Prob.* **19,** 167–182.

WEBER, R. R., VARAIYA, P. AND WALRAND, J. (1986). Scheduling jobs with stochastically ordered processing times on parallel machines to minimize expected flowtime. *J. Appl. Prob.* **23,** 841–847.

WEISS, G. (1988). Branching bandit processes. *Prob. Eng. Inf. Sci.* **2,** 269–278.

WEISS, G. (1990). Approximation results in parallel machines stochastic scheduling. *Ann. Operat. Res.* **26,** 195–242.

WEISS, G. (1992). Turnpike optimality of Smith's rule in parallel machines stochastic scheduling. *Math. Operat. Res.* **17,** 255–270.

WEISS, G. (1995). On almost optimal priority rules for preemptive scheduling of stochastic jobs on parallel machines. *Adv. Appl. Prob.* **27,** 821–839.

WHITTLE, P. (1981). Arm acquiring bandits. *Ann. Prob.* **9,** 284–292.